



Calhoun: The NPS Institutional Archive
DSpace Repository

Theses and Dissertations

1. Thesis and Dissertation Collection, all items

1975-06

Software error detection model

Green, Thomas Frederick

Monterey, California. Naval Postgraduate School

<http://hdl.handle.net/10945/40445>

This publication is a work of the U.S. Government as defined in Title 17, United States Code, Section 101. Copyright protection is not available for this work in the United States.

Downloaded from NPS Archive: Calhoun



Calhoun is the Naval Postgraduate School's public access digital repository for research materials and institutional publications created by the NPS community. Calhoun is named for Professor of Mathematics Guy K. Calhoun, NPS's first appointed -- and published -- scholarly author.

Dudley Knox Library / Naval Postgraduate School
411 Dyer Road / 1 University Circle
Monterey, California USA 93943

<http://www.nps.edu/library>

NAVAL POSTGRADUATE SCHOOL

Monterey, California



THESIS

Software Error Detection Model

by

Thomas Frederick Green

June 1975

Thesis Advisor:

N. F. Schneidewind

Approved for public release; distribution unlimited.

Thesis
G742

NAVAL POSTGRADUATE SCHOOL

Monterey, California



THESIS

Software Error Detection Model

by

Thomas Frederick Green

June 1975

Thesis Advisor:

N. F. Schneidewind

Approved for public release; distribution unlimited.

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Software Error Detection Model		5. TYPE OF REPORT & PERIOD COVERED Master's Thesis; June 1975
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Thomas Frederick Green		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940		12. REPORT DATE June 1975
		13. NUMBER OF PAGES
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) Naval Postgraduate School Monterey, California 93940		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT (of this Report) Approved for public release; distribution unlimited.		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)		
18. SUPPLEMENTARY NOTES		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) Software reliability, error detection, program complexity		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) A model of the error detection process for the testing of software has been developed to investigate the relationship between computer program structure and error detection and test effort. The model has been implemented as a simulation.		

Software Error Detection Model

by

Thomas Frederick Green
Lieutenant, United States Navy
B.A., University of Kansas, 1967

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
June 1975

Author

Approved by:

Thesis Advisor

Second Reader

Chairman, Computer Science Group

Academic Dean

ABSTRACT

A model of the error detection process for the testing of software has been developed to investigate the relationship between computer program structure and error detection and test effort. The model has been implemented as a simulation.

TABLE OF CONTENTS

I.	INTRODUCTION.....	7
II.	NEED FOR RELIABLE SOFTWARE.....	9
	A. SOFTWARE COSTS.....	9
	B. DEFINITIONS.....	11
	1. Terms.....	11
	2. Classification of Errors.....	12
	C. TESTING AND ERROR DETECTION.....	12
III.	ERROR DETECTION MODEL.....	15
	A. NEED FOR A MODEL.....	15
	B. BASIC MODEL DESCRIPTION.....	17
	1. Model Characteristics.....	17
	2. Model Simulation.....	19
	C. MODEL ASSUMPTIONS.....	22
	D. MODEL USES.....	24
IV.	ANALYSIS OF SIMULATION RESULTS.....	26
	A. THE EFFECT OF INCREASING THE NUMBER OF INPUTS..	26
	1. Model Testing.....	26
	2. Simulation Example on a Real Program.....	31
	B. THE EFFECT OF INCREASING THE NUMBER OF ARCS....	33
	C. THE EFFECT OF INCREASING THE NUMBER OF LOOPS...	38
	D. REPLICATING A SINGLE INPUT.....	46
	1. Model Testing.....	46
	2. Simulation Example on a Real Program.....	47
	E. THE EFFECT OF COMPLEXITY.....	47
V.	CONCLUSIONS.....	54
	A. INFERENCES THAT MAY BE DRAWN FROM THE RESULTS..	54
	B. FUTURE WORK.....	55
	FORTRAN EXAMPLE PROGRAM CODE.....	56
	DIRECTIONS FOR USE OF THE ERROR SIMULATION PROGRAM.....	59
	FLOW CHART.....	63

ERROR SIMULATION MODEL COMPUTER PROGRAM CODE.....	112
LIST OF REFERENCES.....	134
INITIAL DISTRIBUTION LIST.....	136

I. INTRODUCTION

Software costs are an increasing fraction of total computer project costs and, for many computer projects, software costs dominate hardware costs. Much of the software development costs are for testing, debugging and integration; a significant part of the costs after releasing the software are for correcting errors. Thus there is current interest in the error characteristics; number, type (overflow, sequence control) and location of software errors in a program. It is generally accepted that computer programs with a complex structure, that is one with a high incidence of branch instructions and loops, are harder to debug and test and more errors persist after release than for programs with a more simple structure. An error simulation model¹ is presented here which investigates the relationship of program structure to error detection and test effort.

Since structure can be controlled during the design phase and measured through all phases of a computer project, the study of the relationship between structure and error characteristics is valuable to the manager of a software project. Complex program structures with poor error characteristics should be avoided. Poor error characteristics result when many errors are located in

¹The suggestion to use a simulation model to study software error detection was given by Dr. Samuel Litwin, a consultant to the Naval Air Development Center.

complex structures in such a way that error detection would prove difficult during testing. In cases where complex program structures may be necessary to help meet program size or speed limitations, it is useful to have an indication of the additional testing which may be caused by complex structures. It is also useful to be able to compare the error characteristics of design alternatives that have different program structures.

II. NEED FOR RELIABLE SOFTWARE

A. SOFTWARE COSTS

The cost of computer systems can be divided into hardware (cost of computers and peripheral equipment) and software (cost to design, write, test and maintain computer programs). The proportion of total costs that is attributable to software has been increasing, projections of higher costs for people and lower costs for hardware components being a continuing trend. The direct cost of software is enormous; one estimate of the cost in the United States, according to Ref. 1, is 10 billion dollars per year. Indirect costs due to late delivery, nonperformance due to errors, wrong actions due to erroneous output, etc., are also very large.

Although there is not extensive data available on the cost of software projects, there is enough data to indicate the magnitude of software costs. References 1 and 2 contain a comprehensive study of Air Force computer costs. The Air Force estimates that software costs in 1972 were from 1 to 1.5 billion dollars, which is 4 to 5 percent of the total Air Force budget, while computer hardware costs were from 300 to 400 million dollars. The percentage of computer costs due to software has increased from 30 percent in the late 1950's to 70 percent now and the estimate for 1985 is 90 percent. The software costs for the U.S. manned space missions from 1960 to 1970 have been estimated at 1 billion

dollars, Ref. 1. In the private sector the same relative magnitudes and trends are present. The development cost of a modern general purpose computer is about one half for hardware and one half for software (operating systems, compilers, support programs). The operating system developed by IBM for the 360 series is estimated to have cost 200 million dollars or about 1000 dollars per instruction in the final version.

The production of software can be divided into three phases:

- * analysis and design,
- * writing programs and
- * test and integration.

Data on how time, effort and money are divided among these three phases gives some indication of why software production is so costly. The fraction of time, effort and money for each phase differs from application to application; however, data from some large projects show similar experience. Estimates are given in Refs. 1 and 2 for some military command and control systems: analysis and design is about 35 percent, writing programs 15 percent and test and integration 50 percent. For space projects the estimates are 35, 20, 45 percent. For the IBM 360 operating system the estimates are 35, 15, 50 percent. Data for business applications indicates less for testing and integration and more for analysis and design than the above data. The surprising amount of time, effort and money for test and integration is often the item most underestimated in planning computer projects, as described in Ref. 3.

B. DEFINITIONS

In the field of software engineering there is little agreement on the definition of terms, such as the definition of software reliability. In order to make the understanding of this paper easier the following definitions will be adhered to as much as possible.

1. Terms

Software reliability is the probability that a computer program will perform its intended function for a specified interval under the stated conditions according to Ref. 4.

Testing is an effort to determine the presence of software errors, not their absence.

Software error is an error in programming logic which leads to undesirable results during program execution.

Module is a particular physical combination of program instructions that is independent of others with respect to compiling, assembling and loading.

Program is a set of modules.

Program complexity may be described by characteristics such as program size, incidence of branch instructions, incidence of loops, incidence of subroutine calls and variety of instructions.

Non-branch instructions may be either computational or input/output instructions.

Structured programming is a programming technique, Ref. 5, in which a program with one entry and one exit can be written using only the following programming progressions:

- * Sequence
- * IF THEN ELSE
- * DO WHILE

Directed graph is a geometric graph, consisting of nodes and arcs, with a direction of traversal associated with each arc.

C. CLASSIFICATION OF ERRORS

Software errors are classified as follows:

- * Mistakes in logic at the flow chart level,
- * Computation and assignment,
- * Sequencing and control,
- * Input/output,
- * Declarations,
- * Key punching/clerical errors committed in writing instructions on coding sheets,
- * New errors introduced as a result of design changes:
 - unexpected side effects caused by changes,
 - logical flaws in change to design,
 - inconsistencies between changed design and implementation,
 - inconsistencies in original and changed hardware

D. TESTING AND ERROR DETECTION

The life cycle of a program is composed of the following phases:

- * Design and analysis,
- * Module development and testing,
- * System integration testing,
- * Functional testing,
- * Maintenance.

The cost of error detection and repair during system integration testing is three times that of testing an individual module during module development testing at TRW, according to Ref. 6. Therefore, the objective should be to reduce the number of errors detected during system integration testing and increase the number (proportion) discovered during module development testing.

In many moderate and large computer projects, a programmer writes and debugs a module and then gives it to a test group. The test group tests the module, integrates it with other modules and then continues testing. The module is tested by supplying an input to the module and then comparing the outcome to the known correct outcome. If there is a mismatch between observed and correct output, an error has been detected. When an error is detected the module is given to a programmer who locates and corrects the error and then returns the module to the test group. Notice the distinction between testing, which is supplying inputs and observing outputs, and debugging, which is the highly individualized detective work needed to locate and correct errors. In debugging, the programmer needs a detailed knowledge of the structure and operation of the module. The tester is frequently unaware of module structure and

operation; he needs only to understand the function of the module.

Most computer programs have a large number of potential inputs; each may exercise a program in a different way. The sequence of instructions of the program that results from a particular input is called the "path" or "thread" associated with that input. Testing by submitting inputs to the program checks only the paths associated with those inputs. For programs with a very large number of inputs, testing can be only a relatively small sampling of all possible inputs, as described in Ref. 3.

III. ERROR DETECTION MODEL

A. NEED FOR A MODEL

Testing is a critical part of software projects because it measures and affects the final quality of the software and it consumes a large part of project time and resources. Testing also reveals the strengths and weaknesses of the analysis, design and coding of the software and gives an estimate of the success or failure of the software after release. Thus it is important to understand the testing process and to understand the relationships between testing and the various decision variables that may be controlled during analysis, design and coding.

A difficult facet of program testing involves the selection of inputs. The tester, who generally is not the person who wrote the code, does not know the specific path that an input will execute. Presently there is no software tool, such as an automatic test data generator, that would allow the tester to force an input to follow a certain path. The closest system designed for testing a certain path is interactive, where the tester selects whichever instruction is to follow the previous one. In this way a particular path is followed, Ref. 7. This is obviously a slow and cumbersome way to check out all, or many, of the possible paths in a program.

Obviously, inputs should be chosen so that a high percentage of the critical paths of the program will be exposed to testing. However, this objective must be weighed against the cost of machine time for debugging and the cost of programming personnel for error correction. A related matter is the determination of when to stop testing. It is usually infeasible to subject a program to all possible input combinations because of resource constraints. Various software packages are available for recording and analyzing the following types of data: count and frequency distribution of types of instructions executed; indication of code which is not executed; and indication of code which is impossible to reach, Ref. 8. Although this type of instrumentation is helpful for tracing program behavior, once a set of inputs is selected, it does not solve the problem of selecting the number and type of inputs in the first place.

Thus, there is a need for a model to examine the relationships between the number of inputs and paths traversed, for a given program structure, and the number of remaining errors, fraction of the program exposed to testing, execution time and repair time. It is of interest to determine the number of inputs required to achieve a specified number of remaining errors for various structures, when the same number of original errors is used with each structure. In addition it is desirable to identify programming structures which have complexities that make it difficult to detect errors.

B. BASIC MODEL DESCRIPTION

1. Model Characteristics

Program complexity may be described by characteristics such as program size, incidence of branch instructions, incidence of loops, incidence of subroutine calls and variety of instructions. Another view of program complexity can be obtained by considering the structure of the program to be a series of nodes, arcs and loops in the form of a directed graph as shown in Figure 1.

In the directed graph used in the simulation model, nodes represent connection points where parts of the program may merge and/or branch and arcs represent a sequence of nonbranching instructions such as computation and input/output. Instructions are located in arcs and errors are located in some of the instructions. An input defines a path from the start node to an exit node. Beginning at the start node an input causes execution of the instructions on its path, consuming test time, until an error is encountered. After the error is thus detected, it is repaired, consuming repair time. There is, however, some risk that the repair will introduce a new error in some instruction. Restarting at the initial node, execution is begun again with the same input. This process is repeated until there are no errors on the path.

Some relative measures of program complexity which are applicable to a directed graph representation of program structure are:

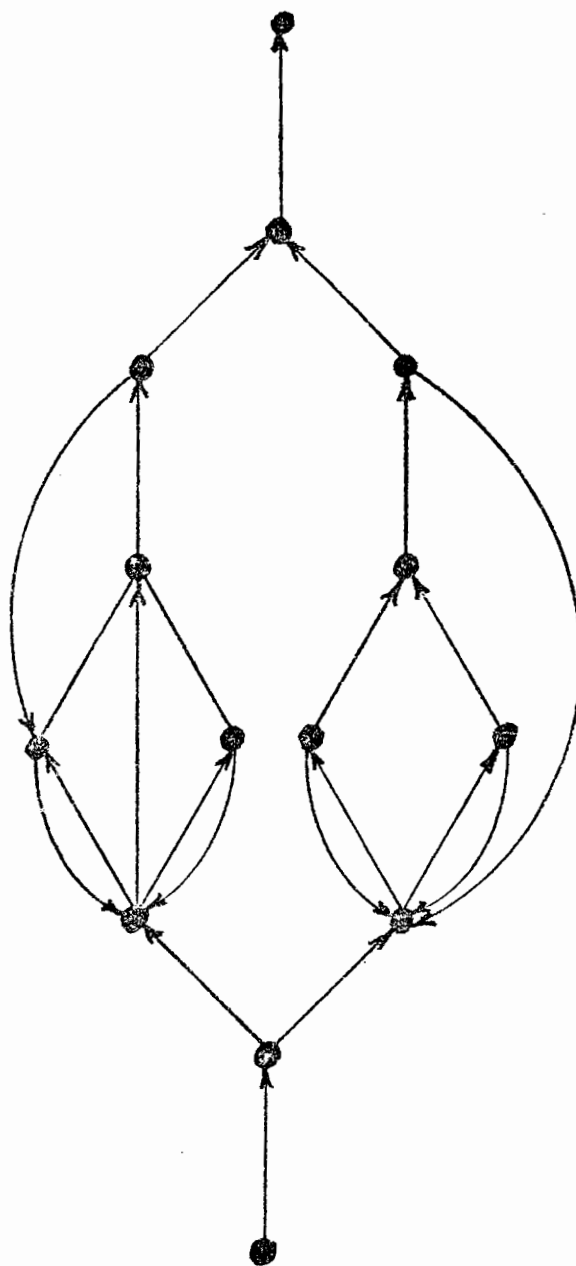


Fig 1 Directed Graph Representation of a Program

- * ratio of actual number of arcs to the maximum possible number of arcs (every node is connected to every other node by an arc),
- * ratio of nodes to arcs,
- * ratio of loops to total number of arcs.

The size of a program is a measure of complexity in an absolute sense. In terms of a directed graph structure, size is determined by the number of nodes, which establishes the number of branch points in a program, and by the number of arcs, which establishes the degree of straightline coding between branch points. Increasing values of the above relative and absolute measures represent increasing program complexity.

2. Model Simulation

The error detection model was written in FORTRAN IV and has been developed and used on the Naval Postgraduate School's IBM 360/67 computer. The program has been executed 40 times in the production mode. The simulation program consists of 639 FORTRAN statements, requiring 194,000 bytes of main memory and executes in 40 to 55 seconds, depending on the type of simulation involved. The directions for use of the error simulation program are listed in Appendix B, Appendix C is the flow chart, which was produced by the FLOWCH routine which is part of the Programming Aids Library at the Naval Postgraduate School, and the FORTRAN code for the simulation model is in Appendix D. The directed graph was input to the simulation as a node-arc incidence matrix. Lacking detailed information about the distributions of the pertinent variables in actual systems, there were no statistical dependencies among the variables established. Thus the random variables were chosen to be independent and

to possess the Markov property. This also makes the model more tractable for obtaining an analytical solution.

The number of instructions per arc is an independent exponential random variable truncated to an integer. Errors are inserted by making the number of instructions between errors an independent exponential random variable, which results in a Poisson distribution of errors per interval of instructions. Errors are inserted by scanning the arcs of the node-arc incidence matrix by columns until the count of instructions from the last error equals the random number.

An input is a sequence of random numbers that determine which arc to traverse at each branch node. For each branch node the probability of taking each arc is equal. This could be changed to test the sensitivity of error detection to different branch probabilities.

The repair times for errors are exponentially distributed. If many programmers work on error repair with each repairing only a small number of errors, the effect of experience on error repair may be small so that a constant repair rate corresponding to the exponential distribution would be appropriate. If few programmers work on repairs, experience would be a factor and an increasing repair rate distribution would be appropriate. For example the log-normal is sometimes used to represent the distribution of hardware repair times, Ref. 8.

The execution times of instructions are exponentially distributed. It was assumed that the execution time of an instruction does not depend on past instruction times. This assumption may not hold if the programmer tends to sequence his instructions in certain patterns.

When errors are repaired, the potential introduction of new errors is simulated. New error insertion is based on the ratio of the number of instructions changed by error repair to the total number of instructions in the arc. The arc where the new error is to be inserted is determined on an equal probability basis.

The simulation is written so that any distribution or its parameter can be changed for the purpose of sensitivity analysis. The choice of distributions may have a significant effect on the simulation results for a given structure; however, since the objective is to evaluate results on a relative basis across various structures, the choice of distributions does not seem to be critical.

For each input, data is collected on the number and location of errors detected, number and location of new errors, number and location of remaining errors, number of arcs traversed, time to execute instructions and time to repair errors.

The simulation model was written so that it would be possible to generate random times for each instruction executed and for each error repaired as the simulation proceeds. However, if the instruction times and repair times are independent and identically distributed as described above, then it is possible and computationally desirable to count the number of instructions executed and the number of errors repaired and multiply these by the average instruction executing time and the average error repair time, respectively, in order to obtain a very good estimate of each total time.

C. MODEL ASSUMPTIONS

A basic assumption of the model is that the tester has some knowledge of the program structure, but that for a given input he does not know the specific path that it will execute. In actual software projects the test group has flow charts and program listings; however, it is infeasible to analyze this information because it may contain thousands of lines of coding. Because of the size of the program, the complicated internal logic and the large number of paths, the relationship between inputs and outcomes is rarely understood. One example is in the testing and maintenance of large operating systems. The relationship of inputs to outcomes is so poorly understood that even after an error has been detected it is often difficult to determine an input that will reproduce the error.

A further assumption of the model is that the tester gains no information as the testing proceeds that will influence his choice of subsequent inputs. In actual software projects the tester should try to make best use of any information gained during testing. Various software packages are available for recording the following types of data count and frequency distributions of instructions executed, indication of code that is not executed and indication of code that is impossible to reach, Ref. 9. However, there are other factors that may make it difficult to effectively use the information gained during testing. For example, the test plan may be specified in advance with no modifications allowed or inputs may be restricted to those that will be typical for the program in actual operation. For these reasons the model assumptions seem reasonable as applied to functional testing.

The probability distributions which were used are listed below.

<u>Property or Event</u> -----	<u>Probability Distribution</u>
* Instructions per arc	Exponential
* Instruction execution time	Exponential
* Original error occurrence	Exponential
* Time to repair an error	Exponential
* Number of instructions affected by repair	Uniform
* New error occurrence	Uniform (based on ratio of instructions changed/ instructions in arc)
* Iterations per loop	Uniform
* Arc selected for new error insertion	Uniform
* Arc selected at branch point for traversal	Uniform

Since little is known about the type of probability distribution which is associated with the above program properties and execution events, the selection of distributions was, of necessity, based on assumptions. However, it was felt that the assumptions were reasonable. For example, the seeding of original errors was based on the number of instructions between errors being exponentially distributed, or equivalently, the presence of an error was independent of the presence of other errors. A second example was that instructions were placed in arcs according to an exponential distribution, or equivalently, the number of instructions between branch points was exponentially distributed. This implies that the number of instructions between two branch points was independent of the number of

instructions between other branch points. Although the choice of distribution may have a significant effect on the simulation results for a given structure, the objective was to evaluate results on a relative basis across the various structures so that choice of distribution was not critical. Although it was possible to vary both the type of distribution and its parameters, the usual procedure was to keep these factors constant and vary program structure, number of inputs and input traversals.

D. MODEL USES

The model can be used to influence software design decisions by making it possible to compare the error detection characteristics of alternative program structures. This is valuable, since error detection characteristics are good indicators of the time and resources consumed by testing. The design flow charts and estimates of branch probabilities and number of instructions can be used to specify programs in the form of a directed graph. The program is then seeded with errors and subjected to random inputs.

The model can also be used to identify the measure or measures of complexity that best predict the ability to detect errors. To do this it is necessary to gather data from the model on the error detection characteristics of a variety of different structures and then do a statistical analysis. This would make it possible to measure the complexity of different programs and then compare the estimates of error detection characteristics. Although some data has been generated, further work is necessary to identify good measures of complexity.

There are other situations where it is useful to be able to compare structures. A frequent problem is to evaluate the cost of adding some additional feature to the program. The results of the model can be used to compare error detection characteristics of the original and modified structure. The problem of how to allocate test effort among structures of different size and complexity can also be addressed.

IV. ANALYSIS OF SIMULATION RESULTS

A. THE EFFECT OF INCREASING THE NUMBER OF INPUTS

1. Model Testing

One would expect that initially there are many errors detected in a program with each input and then the number of errors detected decreases as additional test inputs are used, because much of the program is exposed to testing initially. This is illustrated in Figure 2. The percent residual errors decreased stepwise as the number of inputs increased. In the testing of actual software, after finding many errors, there may be long periods of time with no error detection followed by a new group of detected errors.

Recall that each input in the model detected all the errors in its path, from the input node to one of the output (terminal) nodes. In order to explain the stepwise action in Figure 2, it must be realized that although the paths through the program were, in general, different from previous paths, portions of these paths may have involved only arcs that had been traversed previously. The model had well defined steps where no new arcs were tested for a number of unique input paths, as shown in Figure 3. Thus

30 nodes, 50 arcs, 6 loops
18 original errors, 11 added errors

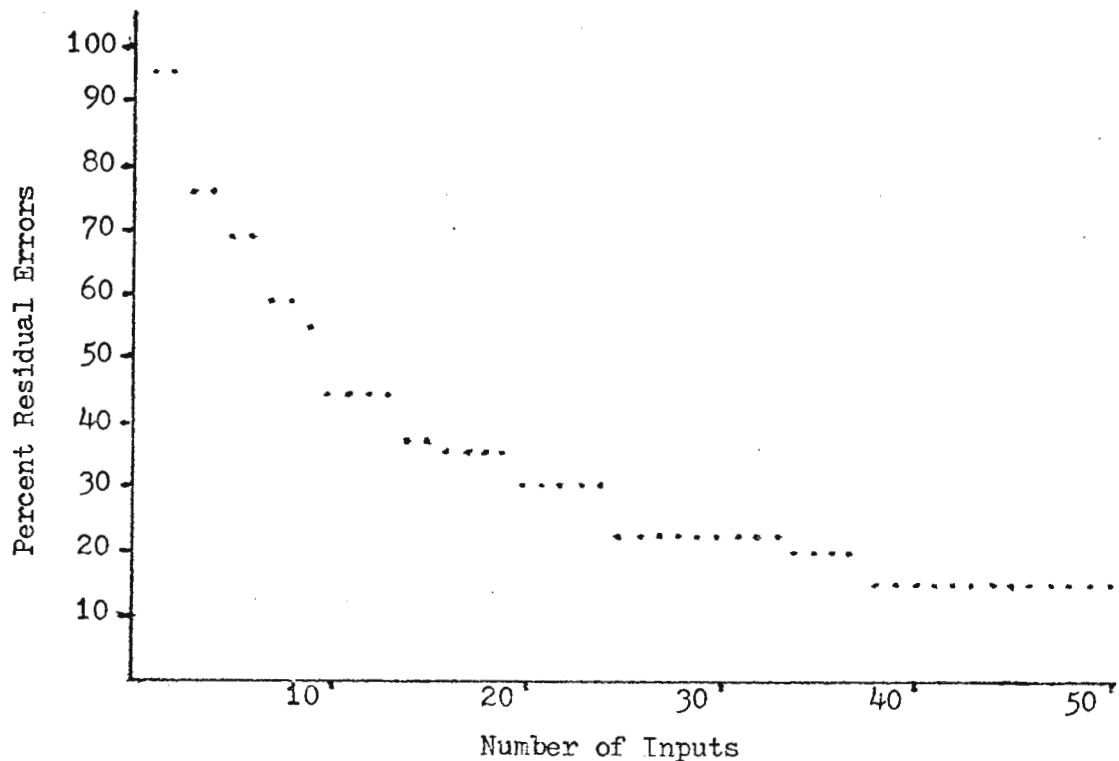


Fig 2 The Effect of Increasing Inputs on Residual Errors

it can be seen how a new group of errors was detected when the model tested previously untested parts of the program. Just because an arc has been previously tested does not imply that it was error free. As each new detected error was repaired, there was some small probability that a new error was introduced in some other portion of the program. This newly inserted error may have been inserted in a previously tested arc. A check was made on the coverage of the arcs by the simulation model. The

30 nodes, 45 arcs, 6 loops
28 original errors, 17 added errors

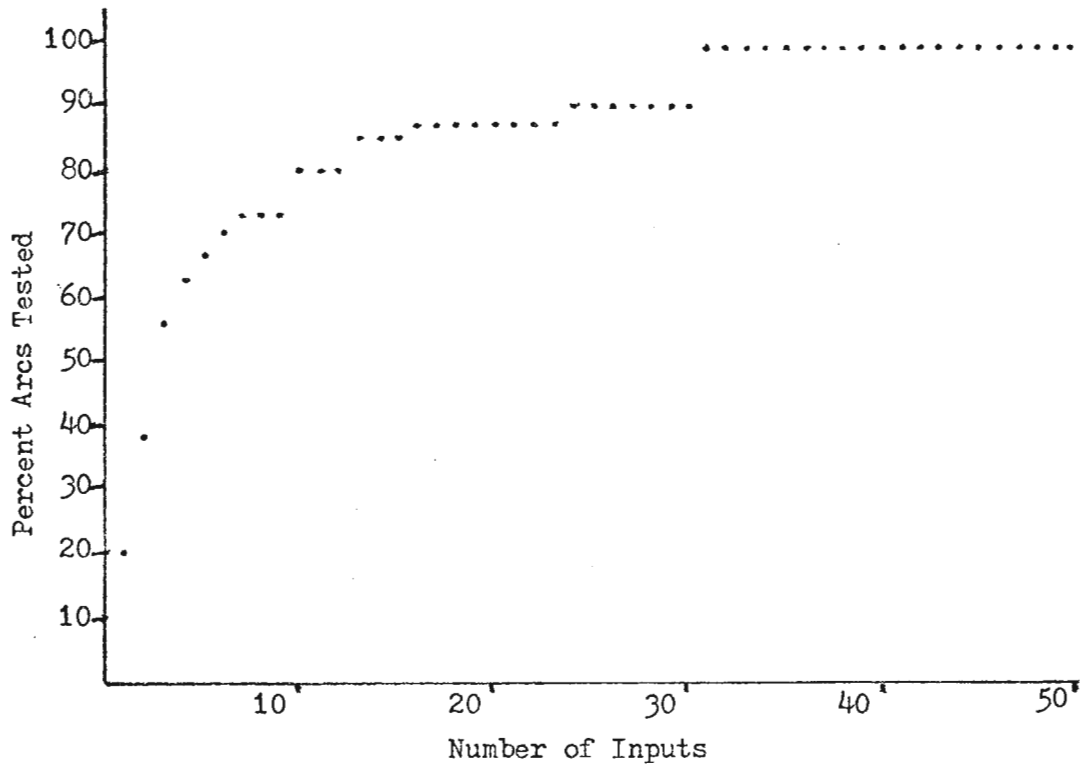


Fig 3 The Effect of Increasing Inputs on Arcs Tested

structure checked had 30 nodes, 40 arcs and 6 loops as shown in Figure 4. The numbers along the arcs indicate the number of times the arc was traversed. For example, the source arc at the top was traversed 50 times, or there were 50 different inputs. Every time an input reached a node it had an equally likely opportunity to select any one of the arcs emanating from the node. The simulation results bear this out as Figure 4 illustrates, where the 50 inputs traversed the top arc and split below with 25 going to the left and 25 going to the right. The arcs which had a backward pointing arc or loop around them were traversed more times as shown

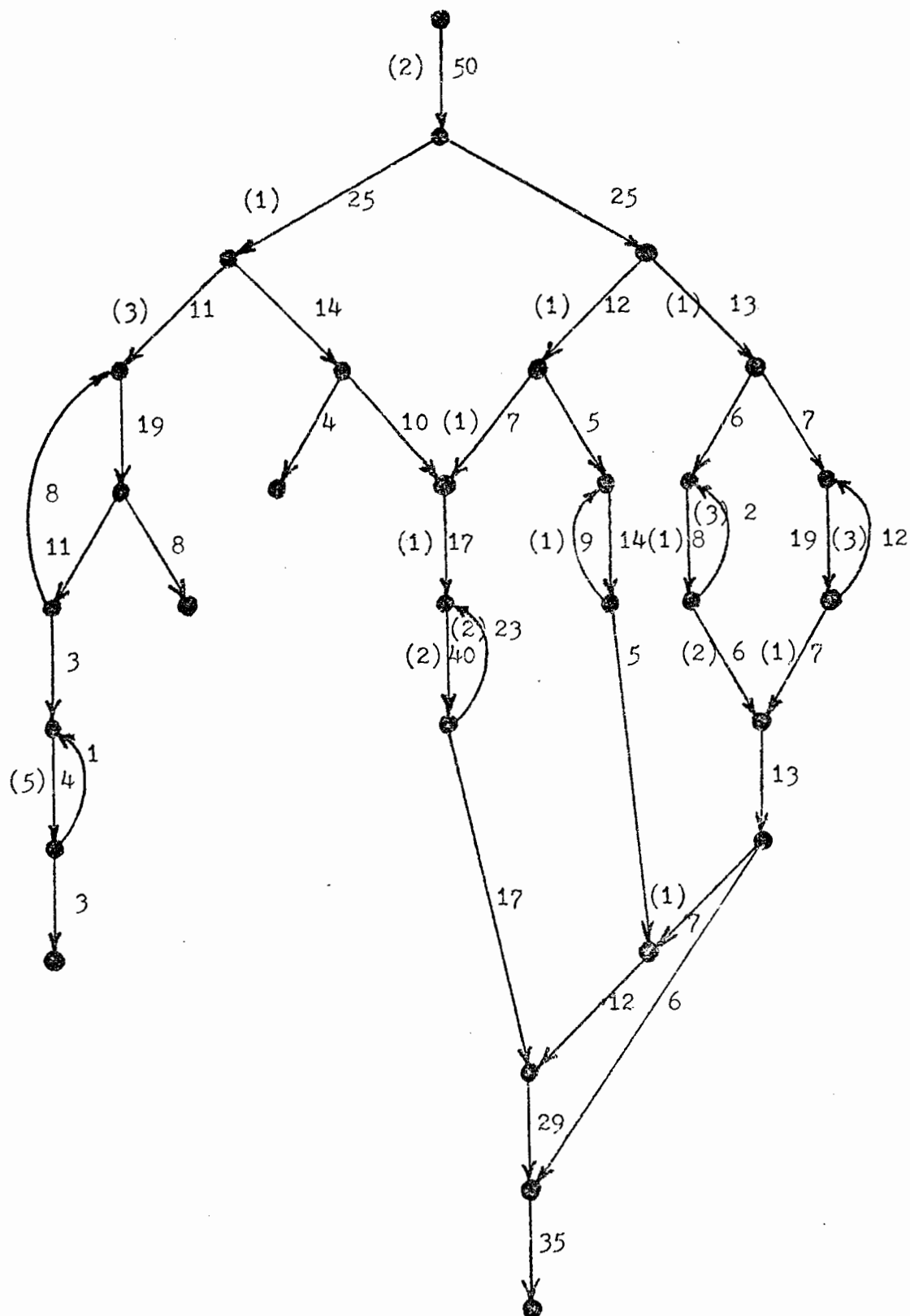


Fig 4 Arc Traversal and Error Patterns

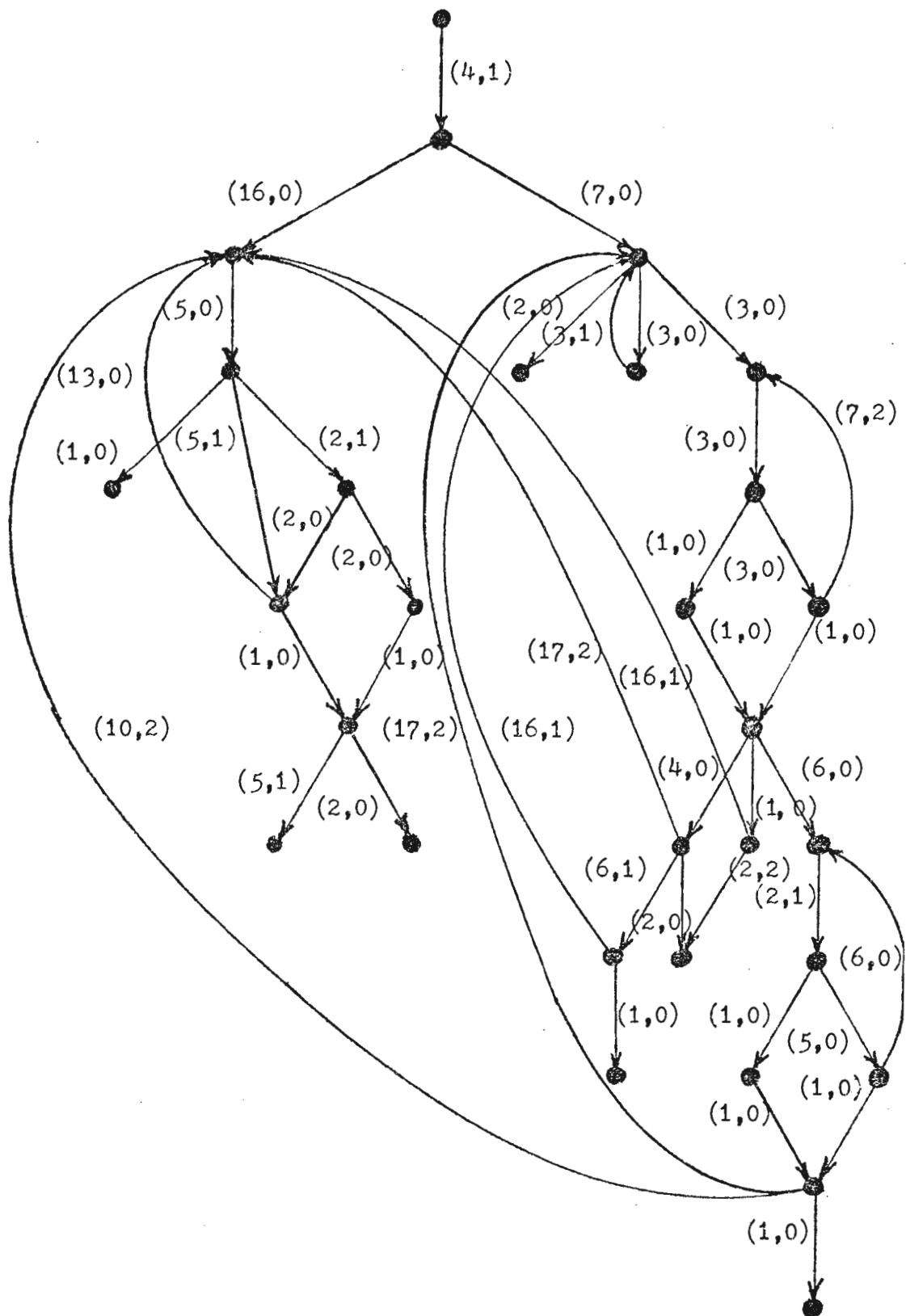


Fig 5 FORTRAN Program Directed Graph

by the number alongside the arc representing the sum of the number of times the backward loop was selected and the number of times the input arc was selected. Looking at the far right hand loop, seven inputs came into the node from above, twelve inputs came into the node from the loop and 19 of the inputs exited the node. The parenthesized numbers indicate the number and location of errors. The number of errors includes the errors initially seeded and the errors inserted when repairing detected errors.

2. Simulation Example on a Real Program

In order for the simulation model to be of any practical use, it had to be tested on a real program. Appendix A contains the code and structure of a textbook FORTRAN program for computing Bessel Functions. The column labeled "node" corresponds to the nodes in the directed graph representation of the program in Figure 5. This particular program was selected as an example of a good computational program, since it was presented in a numerical analysis text, Ref. 10, and an example of a poorly coded program, since a casual reading of the code showed a lack of use of structured programming techniques. Another reason for the selection was that the program could be broken down into 30 nodes, which was the same as the test structures. It also fit within the range of structures tested having 43 arcs and 9 loops. The first number inside of the parenthesis represents the number of instructions in the arc and the second number represents the number of errors, original errors plus added errors, in the arc.

Fifty randomly selected inputs were run through the structure. With 16 errors initially seeded, five errors, or 16.7 percent of the total errors seeded, still remained

after fifty inputs. Comparing this result with a test structure with 45 arcs and 6 loops and another test structure with 44 arcs and 10 loops, the percent residual errors in the FORTRAN program was high, illustrated by Figure 6. By analyzing the paths each input traversed it was noted that six of the nine loops in the FORTRAN program, all emanating from the bottom of the graph, going to the top of the graph, were very seldom used. Thus some individual inputs were not given an opportunity to loop back up to the top of the graph, and thus test more branches for a given input.

FORTRAN Program
30 nodes, 43 arcs, 9 loops
16 original errors, 14 added errors

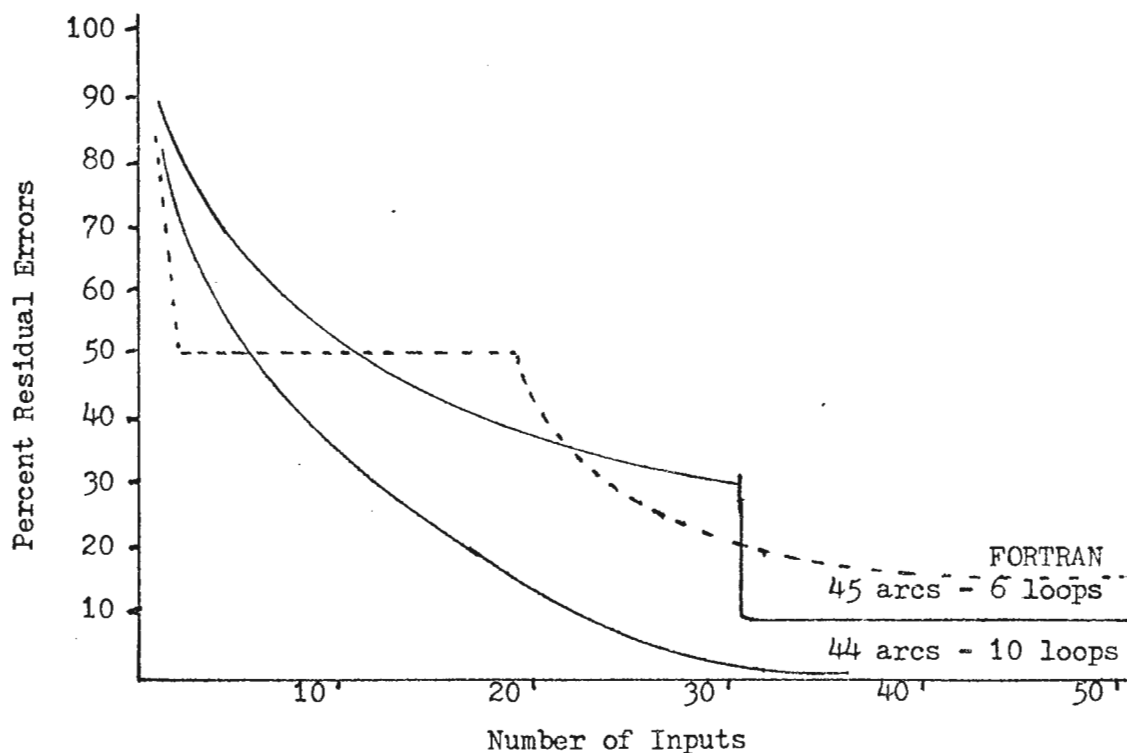


Fig 6 Residual Error Pattern

This was borne out by the results in Figure 7, which showed that the percentage of arcs tested was lower for the FORTRAN program than for the other structures tested.

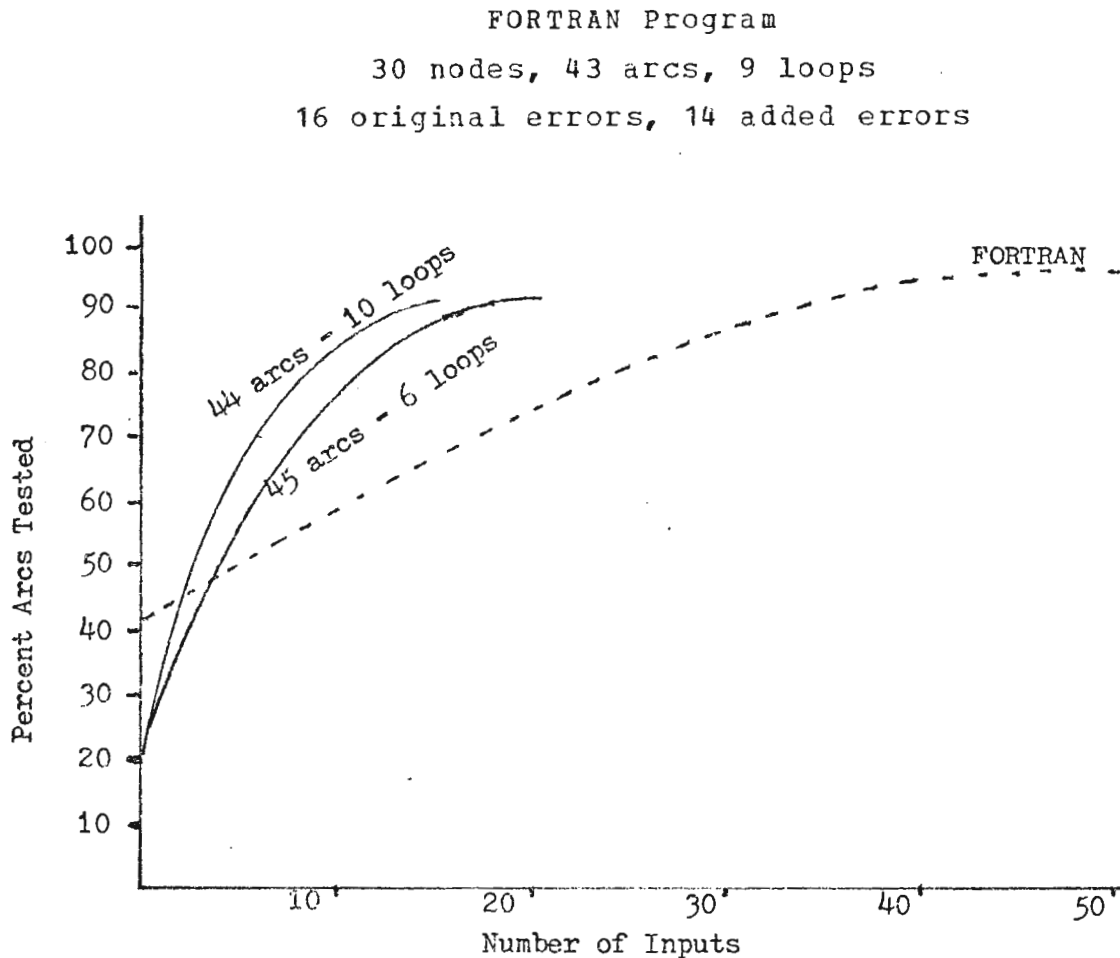


Fig 7 Arcs Tested Pattern

B. THE EFFECT OF INCREASING THE NUMBER OF ARCS

Intuitively, given two programs with the same number of nodes, and a different number of arcs emanating from the

nodes, one would expect that the program with the greater number of arcs, or the more complex program, would have the higher percentage of residual errors. By the same reasoning one would expect the more complex program to have fewer arcs tested with a given number of inputs.

Fifty random inputs were used on each of the program structures described below. Each structure contained thirty nodes and six loops. Retaining the concept that each node represents a branch or decision point in the program, the most simple structure that can be defined, using thirty nodes, must have a minimum of forty arcs. By definition, to establish a node there must be at least three arcs, in any combination, either terminating or emanating from the node. Thus, the minimum number of arcs in a structure is $3/2$ (the total number of nodes minus the number of entry and terminal nodes). Recall that an arc was defined as either a forward or backward pointing arc, called a loop. Starting with forty arcs and adding five more to each structure, five structures were simulated with 40, 45, 50, 55 and 60 arcs. After fifty inputs the percent residual errors increased as the number of arcs increased, as Figure 8 illustrates. The percent residual errors was chosen as the vertical axis in Figure 8 rather than residual errors since the number of errors, original errors plus added errors, varied in each of the five structures. The reason for the variable number of errors was that each time a new structure was defined, the error simulation program would randomly seed all the original errors again, thus errors could have been inserted into the added arcs.

30 nodes, 40 - 60 arcs, 6 loops

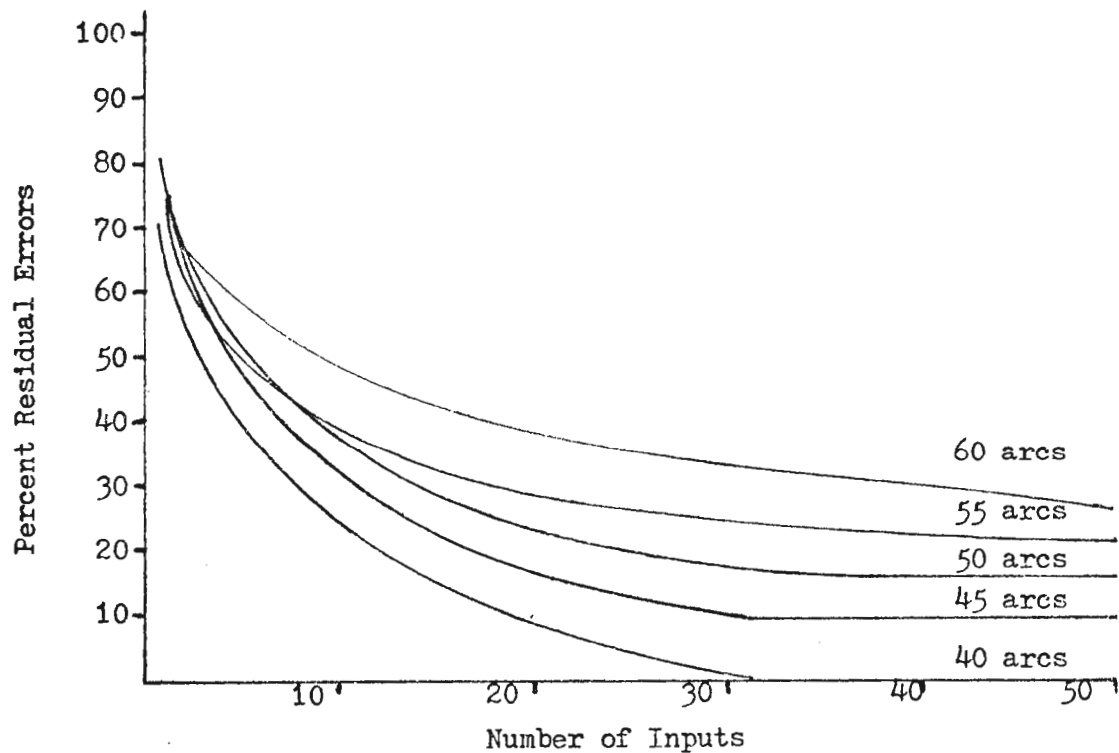


Fig 8 Relation Between Complexity and Residual Errors

Similarly, Figure 9 illustrates the effect of increased complexity on the percentage of the arcs tested. As the number of arcs increased, the percent of the arcs tested decreased.

Examining the paths traversed by each input gave some insight as to why an increased number of arcs caused higher residual error and lower percentages for arcs tested. When an arc was added, the number of arcs emanating from a node increased. There was a positive probability that the added arc could contain an error as the entire structure was seeded with errors anew. As the number of arcs increased, there were also more arcs which provided shorter paths to an exit node

30 nodes, 40 - 60 arcs, 6 loops

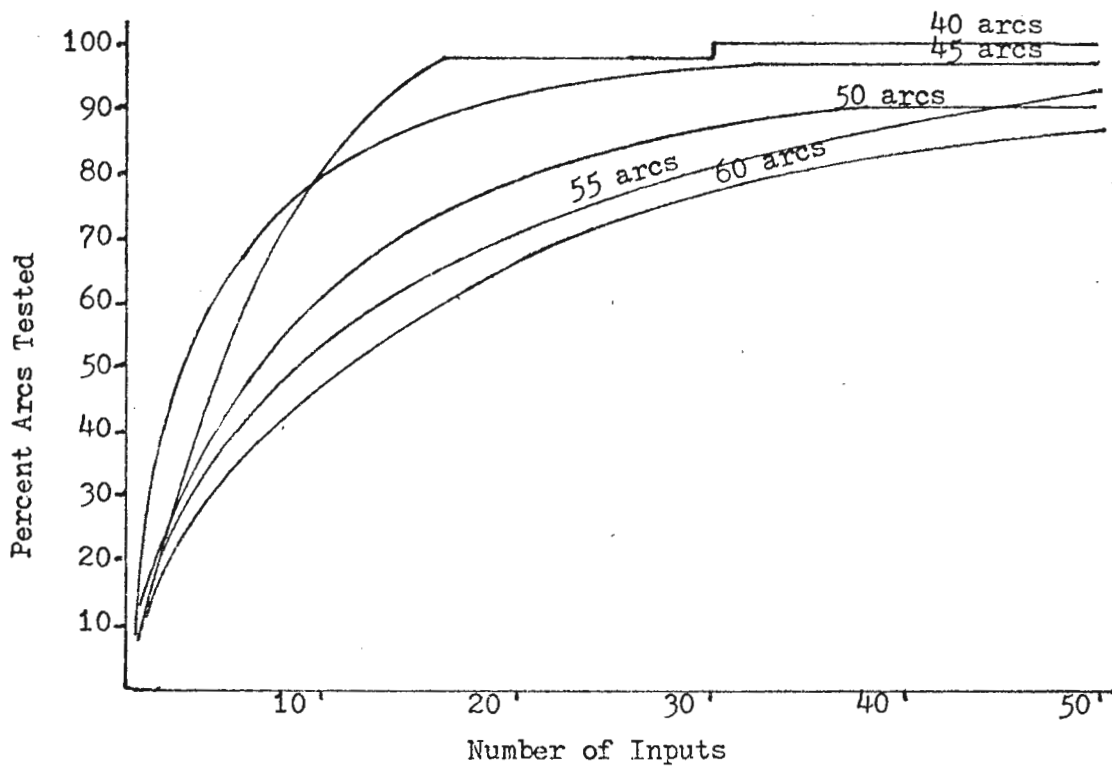


Fig 9 Relation Between Complexity and Percent Arcs Tested

by connecting a node closer to the input with a node closer to one of the outputs, thus leaving some intermediate arcs untested.

Repair time turned out to be unrelated to complexity. The number of errors initially seeded controlled the repair time. These results can be seen in Figure 10, where the data points have been smoothed to give straight lines. Several structures are shown in the plot of repair time versus percent residual errors. The amount of time required to repair errors, for a given percentage of residual errors, increased as the number of errors initially seeded increased.

30 nodes, 40 - 60 arcs, 6 loops

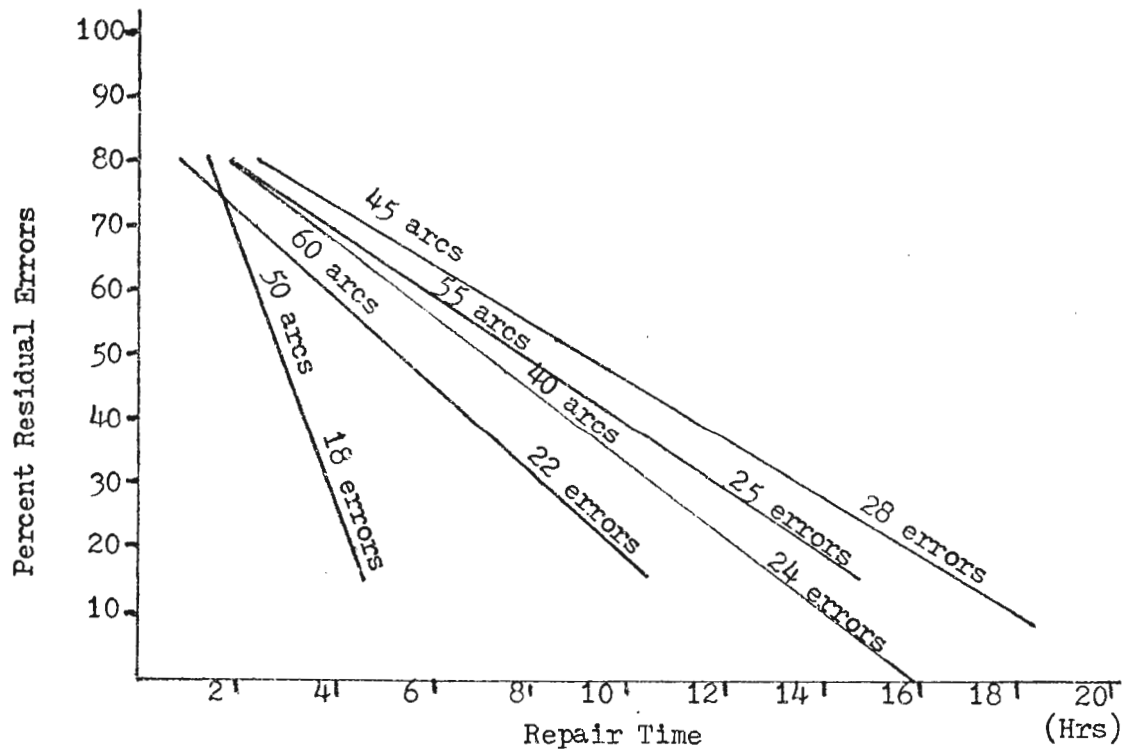


Fig 10 The Effect of Arcs on Repair Time

Generally the relationship between the percent arcs tested and the percent residual errors can be described as approximately linear. As the percentage of the arcs tested increased, the percentage of the residual errors remaining decreased. In Figure 11, the shaded area represents the band of values, corresponding to various structures, for a given percentage of arcs tested. The solid curve is the mean value.

30 nodes, 40 - 60 arcs, 6 loops

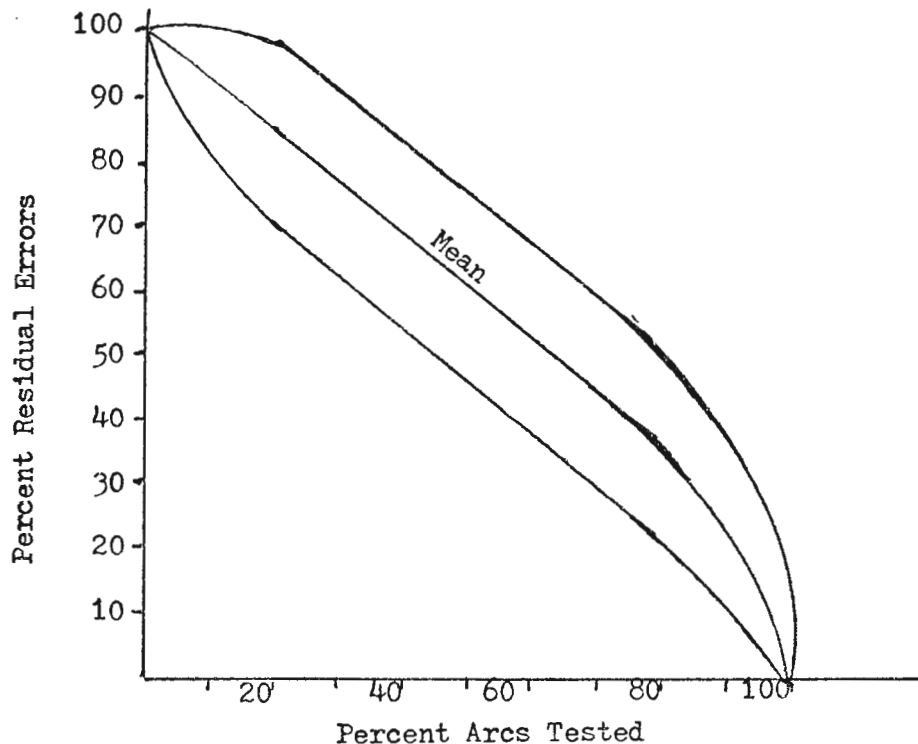


Fig 11 The effect of Complexity on the Relationship between the Residual Errors and Arcs tested

C. THE EFFECT OF INCREASING THE NUMBER OF LOOPS

Improper loop indexing is usually near the top of a list of most frequently occurring errors, Ref. 1. Many people think that loops should be eliminated as a program structure. Note that the only influence that loops play in this model is with respect to coverage. The model does not account for errors in the loop counter or failure to get out of a loop. One of the results of the analysis, as shown in Figure 12, was that an increase in the number of loops had no significant effect on the percentage of residual errors.

Structures with zero, 5, 6, 10, 14 and 20 loops were analyzed using the error simulation program. The percent residual errors was chosen as the vertical axis rather than residual errors since the number of errors, original errors plus added errors, varied in each of the six structures. The reason for the variable number of errors was that each time a new structure was defined the error simulation program would randomly seed all the original errors again, thus errors could have been inserted into the added loops.

30 nodes, 34 - 54 arcs, 0 - 20 loops

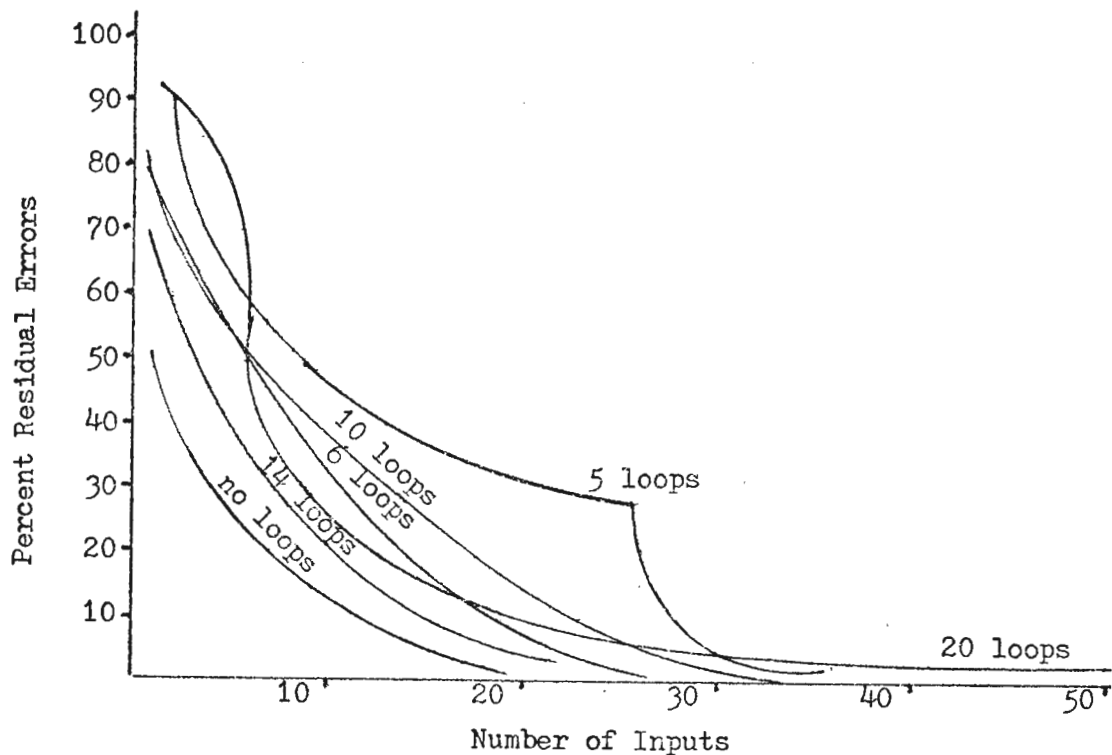


Fig 12 The Effect of Loops on Residual Errors

The reason for the independence of the percent residual errors from loops can probably be explained by the fact that there was no distinguishable difference between the percent

of arcs tested in the six cases with 0, 5, 6, 10, 14 and 20 loops. This is shown in Figure 13.

30 nodes, 34 - 54 arcs, 0 - 20 loops

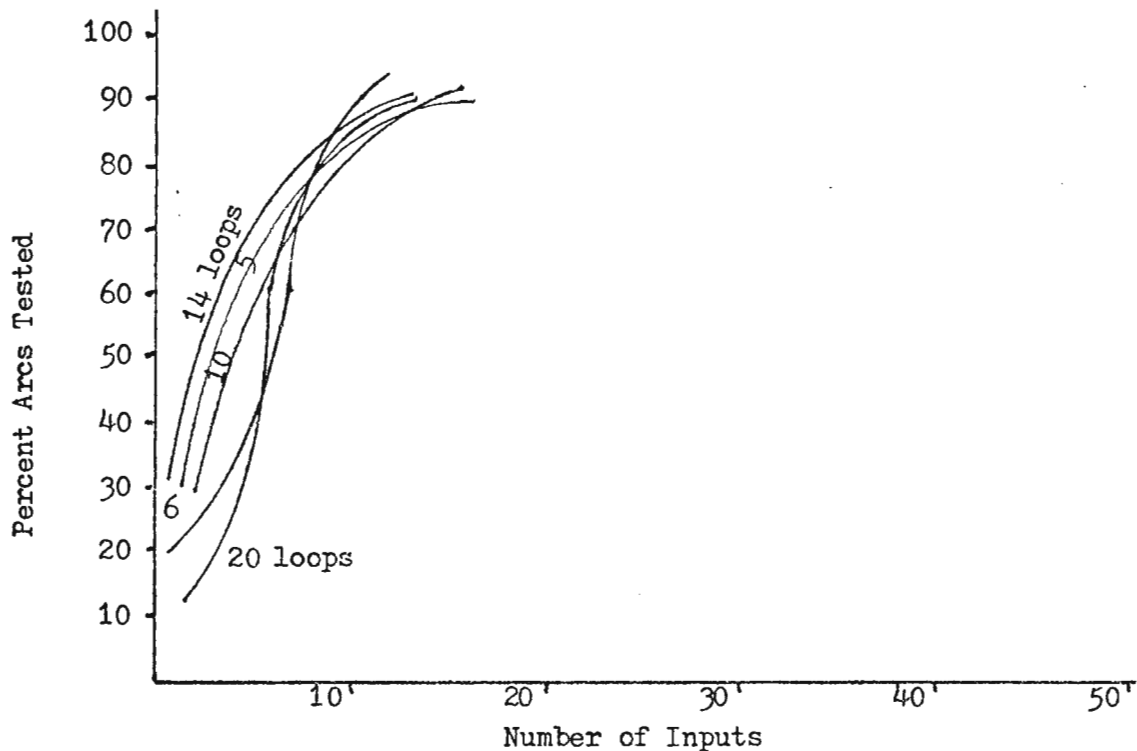


Fig 13 The Effect of Loops on Percent of Arcs Tested

By examining the paths the inputs trace, the explanation of the above becomes obvious. After an input completes a loop, it once again has an opportunity of branching out of the loop, thus testing more arcs than a structure with no loops. Each time another loop was added, the probability of branching out of all the loops increased at approximately the same rate as the increased number of loops. This concept was reinforced by the data shown in Figure 14. After the structure was expanded to nine loops, additional loops had no effect on error detection. The percent of the

total residual errors in the structure that resided in the loops was a constant 59 percent and the percent in the arcs was a constant 41 percent for structures with nine to twenty loops. This data was derived by starting with a structure containing 20 loops, seeding errors, and then analyzing the structure. The error simulation program would then delete a loop and its associated error, if one had been seeded, and then analyze the new structure.

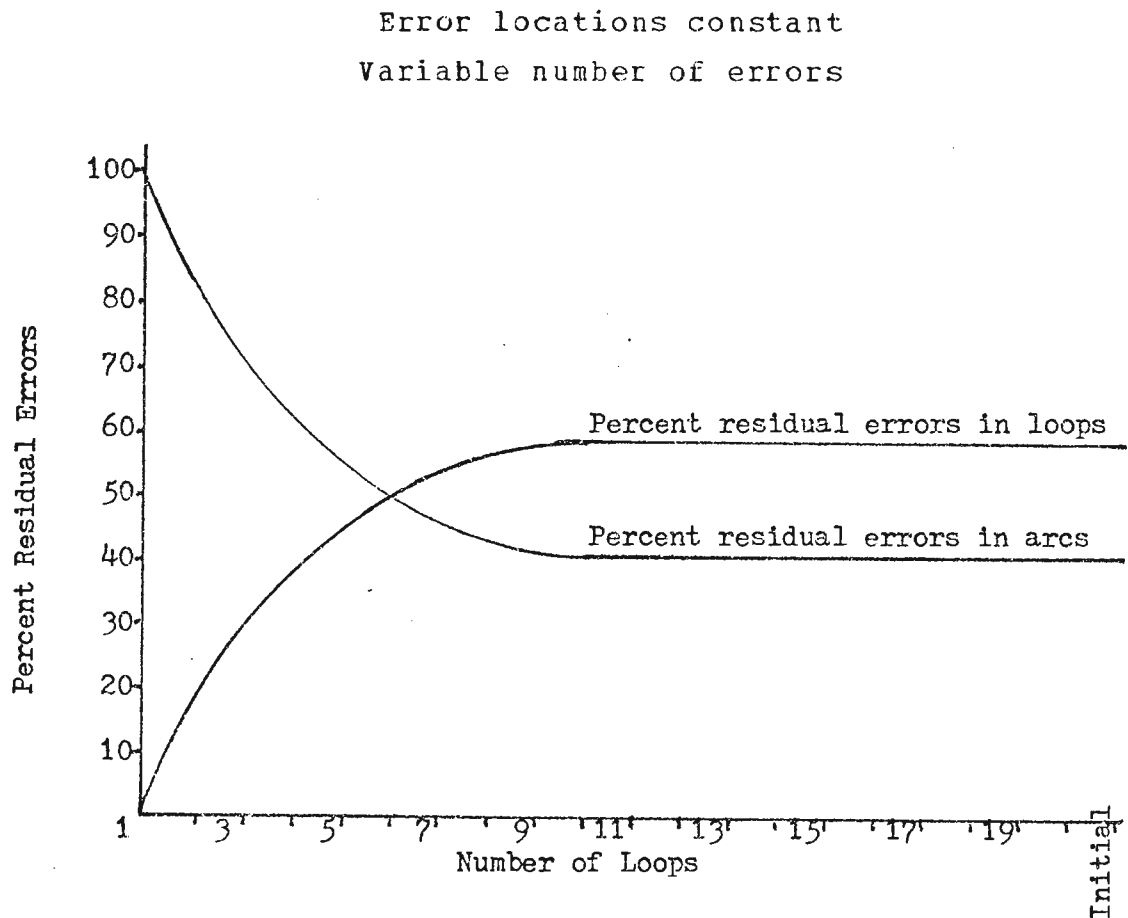


Fig 14 Breakdown of Residual Errors in Loops and in Arcs

It required essentially the same amount of repair time to decrease the percent residual errors to a certain level for all the structures containing loops, as shown in

Figure 15. The structure with no loops took less than half the repair time to get to the same percent of residual errors as a structure with loops, for a percent residual errors less than 50.

30 nodes, 34 - 54 arcs, 0 - 20 loops

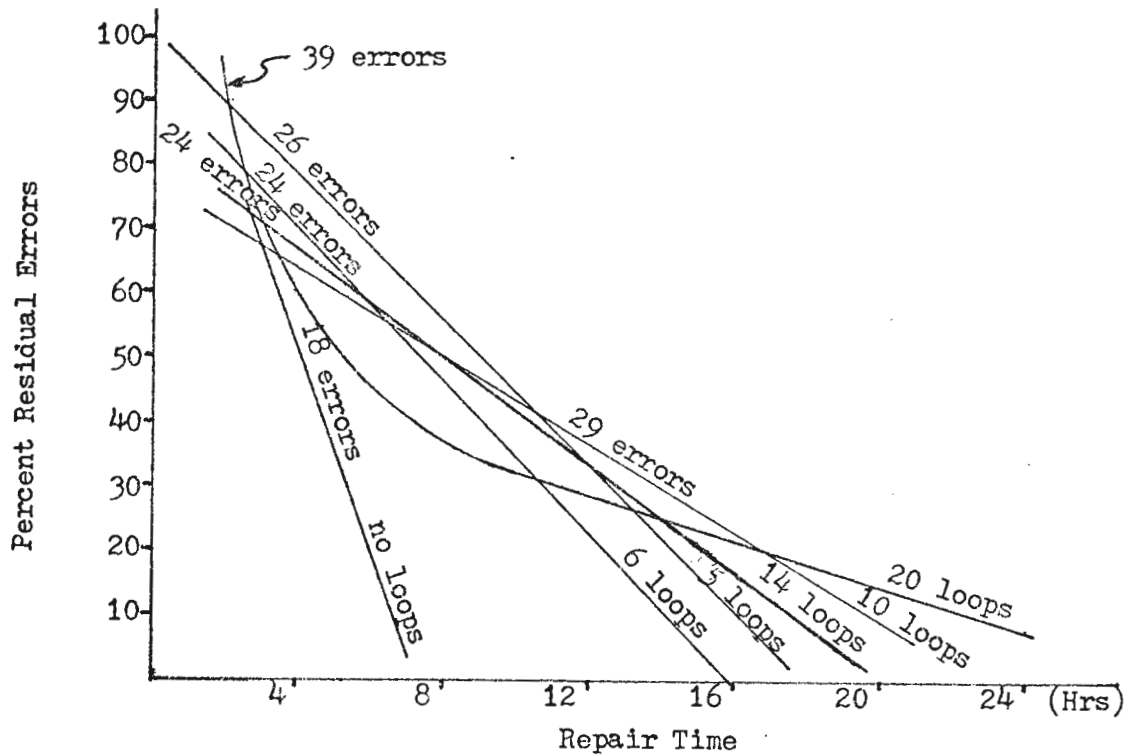


Fig 15 The Effect of Loops on Repair Time

In Figure 10 it was shown that the number of errors initially seeded determined repair time. In Figure 15, the structure with no loops had fewer initially seeded errors than all the structures with loops, so the above relationship holds for this case. However, the number of errors initially seeded had no distinguishable effect on the repair time of the structures with loops. Thus, the repair

time must be a function of both the number of errors initially seeded and whether or not loops are present.

It was not possible to make any judgements concerning the determinants of execution time. This was due to the fact that all but one structure tested had loops. Loops were executed a variable number of times as determined by a uniform distribution which established the number of iterations. The effect of a doubly nested DO loop was captured by allowing an input to have an equiprobable chance of branching back up to the start of the loop or of branching farther down the structure. The relationship

30 nodes, 34 arcs, no loops
18 original errors, 6 added errors

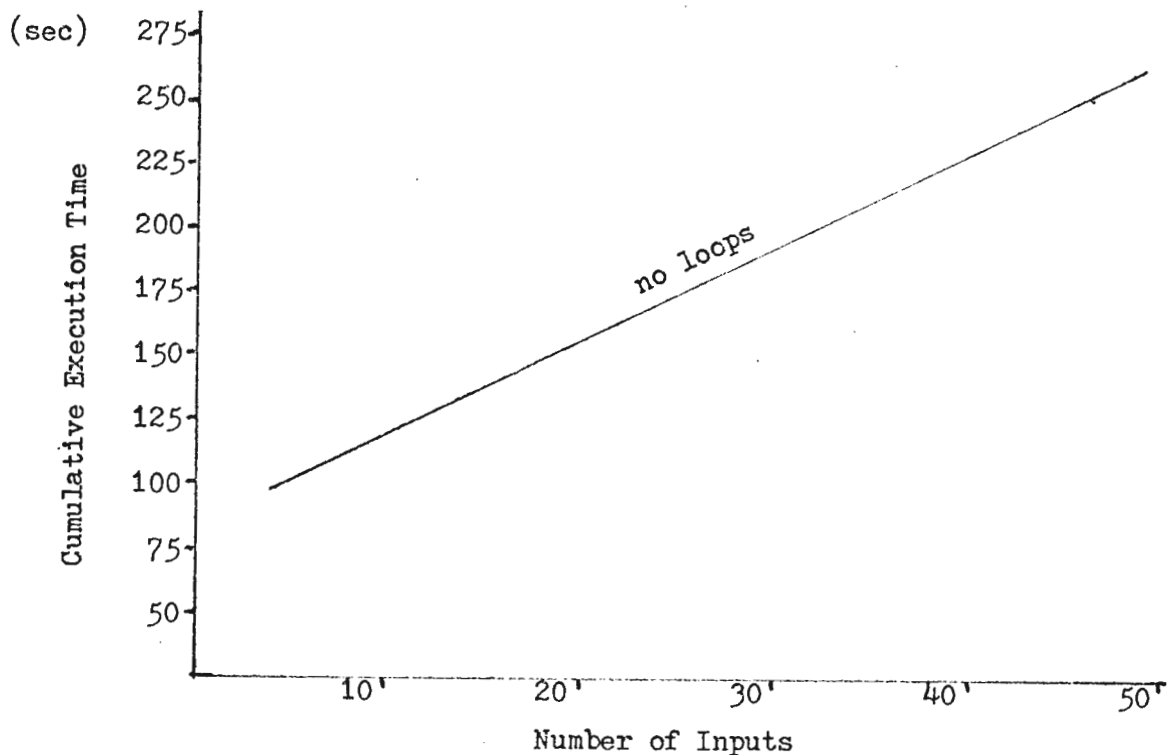


Fig 16 Execution Time for a Structure with No Loops

between the number of inputs and the cumulative execution time for a structure with no loops is examined in Figure 16. A plot for any of the structures with loops has points scattered all over due to the random effect of the loops on execution time.

The effect of the percentage arcs tested, with loops present, on the percent residual errors can loosely be described as a linear relationship. As long as the structures all had loops, the curves of the percentage of

30 nodes, 34 - 54 arcs, 0 - 20 loops

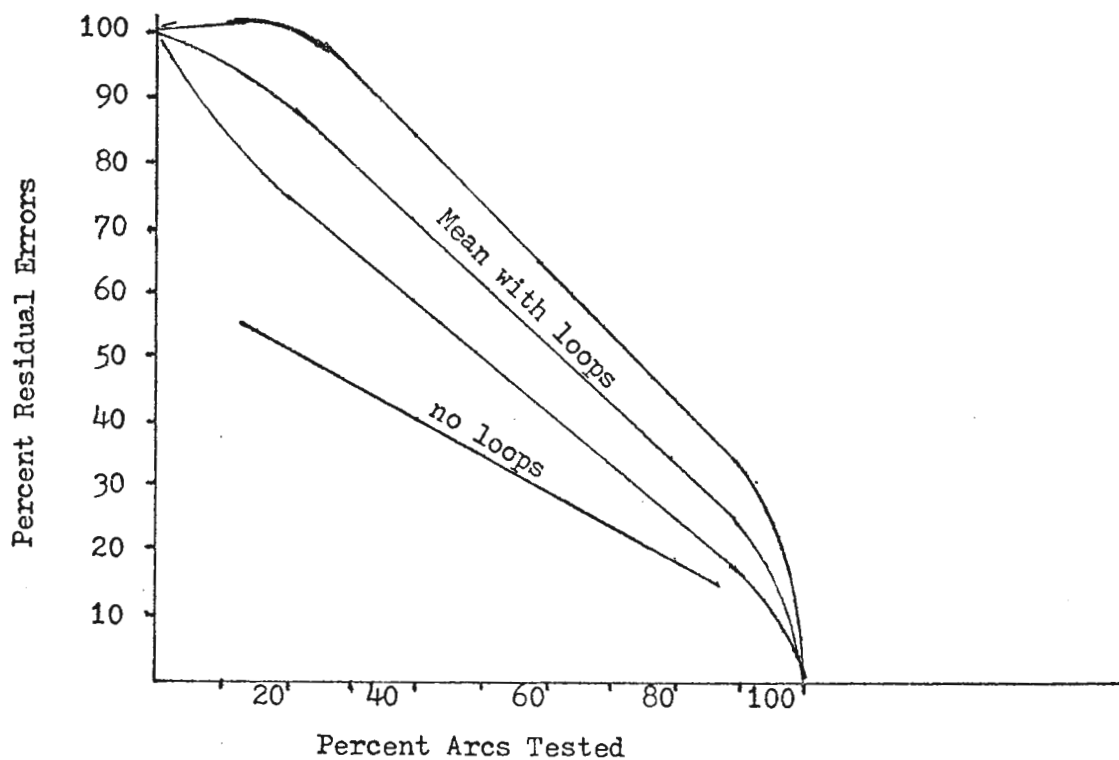


Fig 17 The Effect of Loops on the Relationship between Residual Errors and Arcs Tested

arcs tested versus the percent residual errors all fall within a narrow band of values as shown by the shaded area of Figure 17. The curve for a structure with no loops is also plotted.

The mean of the percent residual errors of structures with a variable number of arcs and a constant number of loops and the mean of percent residual errors of structures with a variable number of loops are plotted in Figure 18, which shows that the two curves are almost identical. However, as Figure 18 illustrates, the structure with no loops required significantly fewer arcs to be tested to achieve the same level of residual error percentage as compared to the structures with loops.

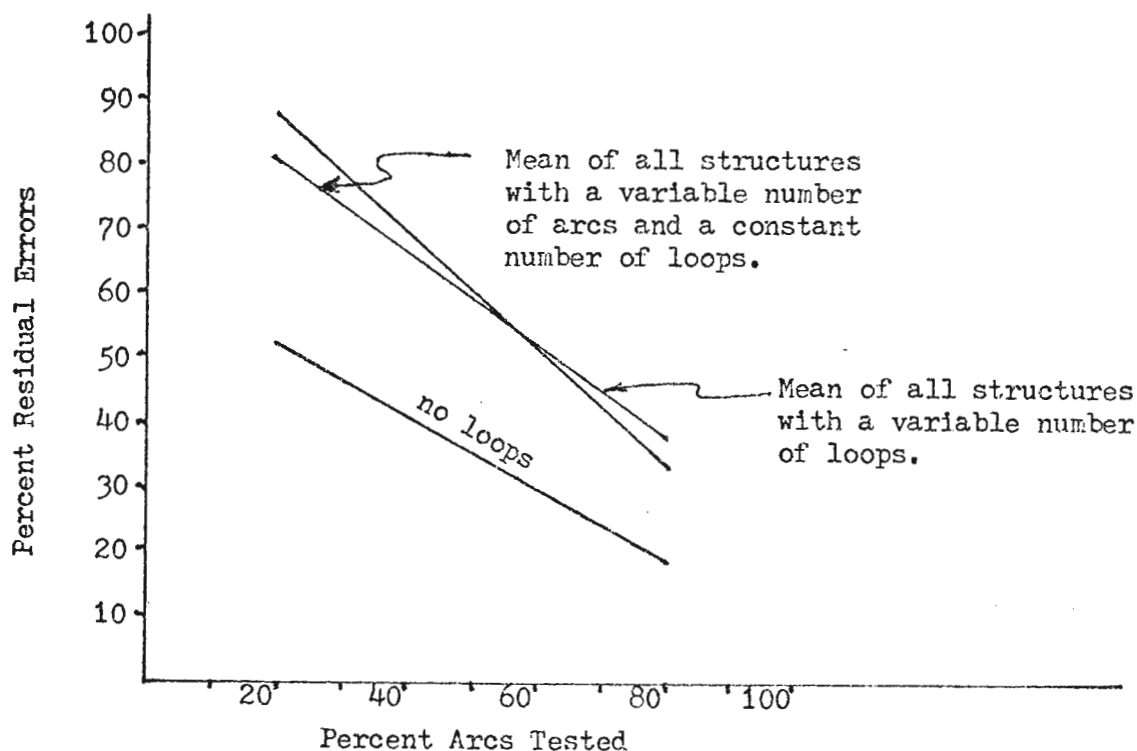


Fig 18 The Advantage of No Loops

D. REPLICATING A SINGLE INPUT

1. Model Testing

The usefulness of the model is now examined for predicting the ability of detecting errors in an actual program. Four pieces of information are of importance for a manager conducting module development testing of computer programs. These are: the percent or number of residual errors, the percent or number of arcs tested, the amount of repair time, and the amount of execution time.

The simulation model was used on ten different test structures to see if this information could be predicted. For each structure a single randomly selected input was run and the above data was collected. This process was replicated 100 times, or in other words, 100 randomly selected inputs were used with each input using the same structure and the same number of errors seeded in the same places. Statistics such as mean, median, variance, standard deviation, etc., were calculated.

As an example, the basic 30 node, 40 arc, 6 loop structure had 24 errors initially seeded. The simulation model produced a mean of 78.79 percent residual errors with a standard deviation of 9.10 for one input. Thus, one could estimate that based on 24 original errors, 78.79 percent of the errors will remain after one input. Similar statistics were determined for percentage of arcs tested and repair time. Execution time was found to have a high variance. For instance, the mean execution time for one input for the above structure was 32.50 seconds with a standard deviation

of 50.15. Thus, estimates of execution time based on the mean would be subject to high error.

2. Simulation Example on a Real Program

The simulation model was used on the FORTRAN Bessel Function program described earlier. It was found that, based on 16 original errors the expected percent residual errors was 84.26 percent with a standard deviation of 9.09, or 15.74 percent of the original errors could be expected to be found and corrected with one input. Similarly, 17.70 percent, with a standard deviation of 8.65, of the arcs could be expected to be tested by one input. Of prime importance to the project manager, 1.41 hours of repair time, with a standard deviation of 1.18, a relative measure for the manager to use when comparing alternative structures, could be expected to be devoted to detecting and repairing 15.74 percent of the errors.

E. THE EFFECT OF COMPLEXITY

The following complexity measures will be used:

- * AMA is the ratio of the number of arcs in the structure to the maximum number of arcs possible for the given number of nodes,

- * NA is the ratio of the number of nodes to the number of arcs in the structure,

- * LA is the ratio of the number of loops to the number of arcs in the structure.

Using these relative complexity measures, it was of interest to see how each of the measures affected the percent residual errors and the percentage of arcs tested.

Five different structures with a constant number of loops and a varying number of arcs and six different structures with a varying number of loops were examined. For each structure, 100 replications of a single input were simulated using the error simulation program, and statistics were gathered about the percent residual errors and the percentage of arcs tested.

In Figure 19, the percent residual errors after one input increased as the complexity increased. In this case the complexity measure was the ratio of the actual number of arcs to the maximum number of arcs possible with a given number of nodes. Similarly, using the same complexity measure the percent arcs tested after one input decreased as the complexity increased, as shown in Figure 20. In both Figures 19 and 20, the standard deviation from the mean,

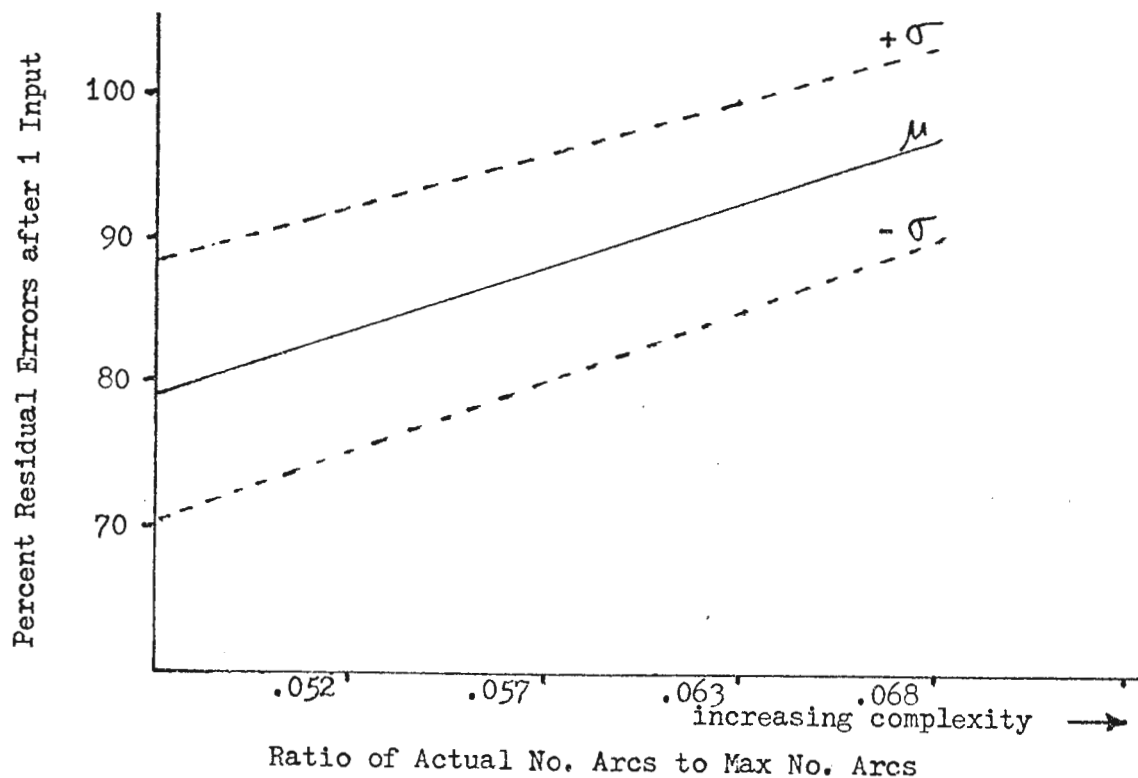


Fig 19 The Effect of AMA on Residual Errors

represented by the dashed lines, decreased as complexity increased.

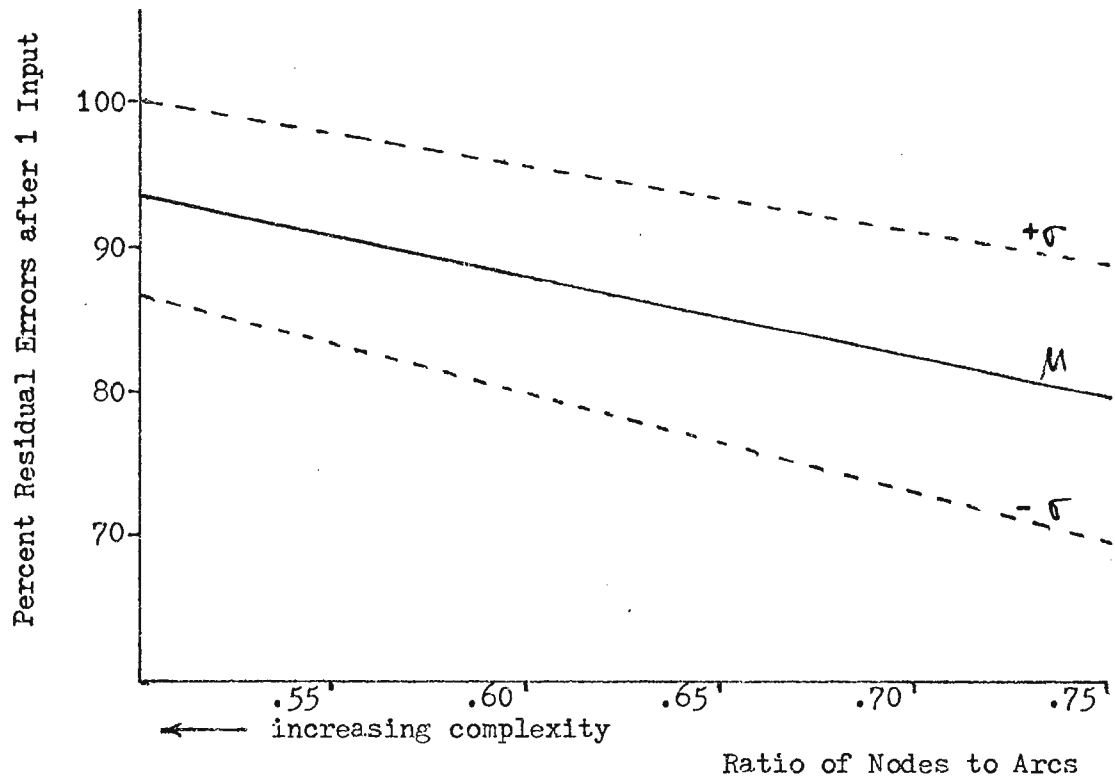


Fig 20 The Effect of AMA on Arcs Tested

Using the ratio of nodes to arcs as a complexity measure similar results were obtained. The percent residual errors increased and the percentage of the arcs tested decreased as the complexity increased. These results can be seen in Figures 21 and 22, where increasing complexity is from right to left. Note that there was an even sharper decrease in the standard deviation as complexity increased in both Figures 21 and 22.

Constant number of nodes
Variable number of arcs

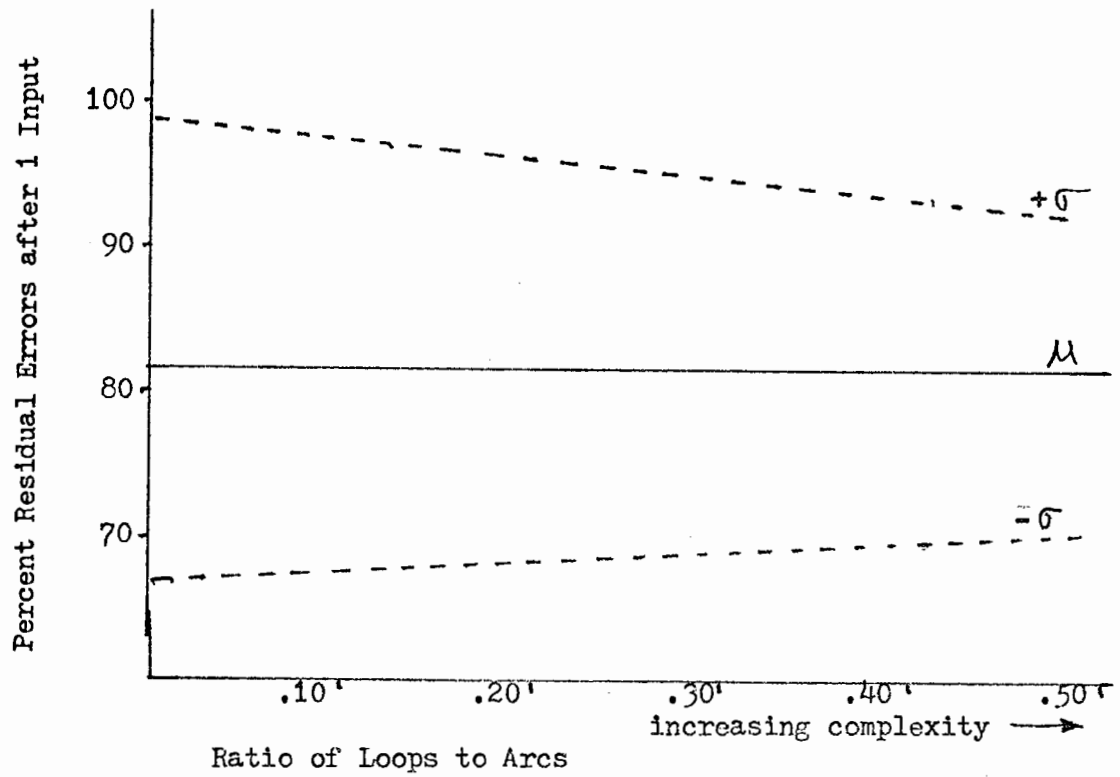


Fig 21 The Effect of NA on Residual Errors

Constant number of nodes
Variable number of arcs

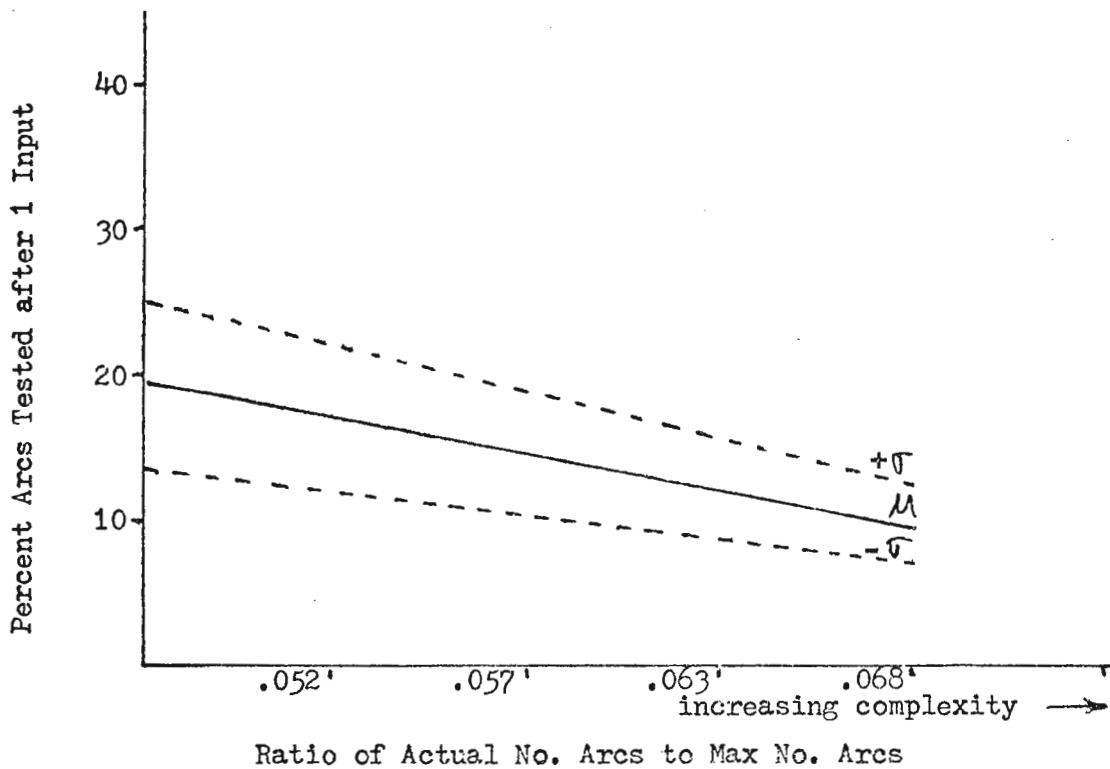


Fig 22 The Effect of NA on Arcs Tested

A third complexity measure used was the ratio of loops to arcs. In Figure 23, the percent residual errors remained constant for one input as the complexity increased. Once again this reinforced the idea that loops expose more arcs to testing at the same rate as the additional arcs increase the complexity. In Figure 24, it can be seen that the added complexity had no effect on the percentage of the arcs tested after one input, with the mean after 100 replications being a constant 22 percent. Since loops were also defined as arcs, the ratio of loops to arcs did not increase linearly as the number of loops increased.

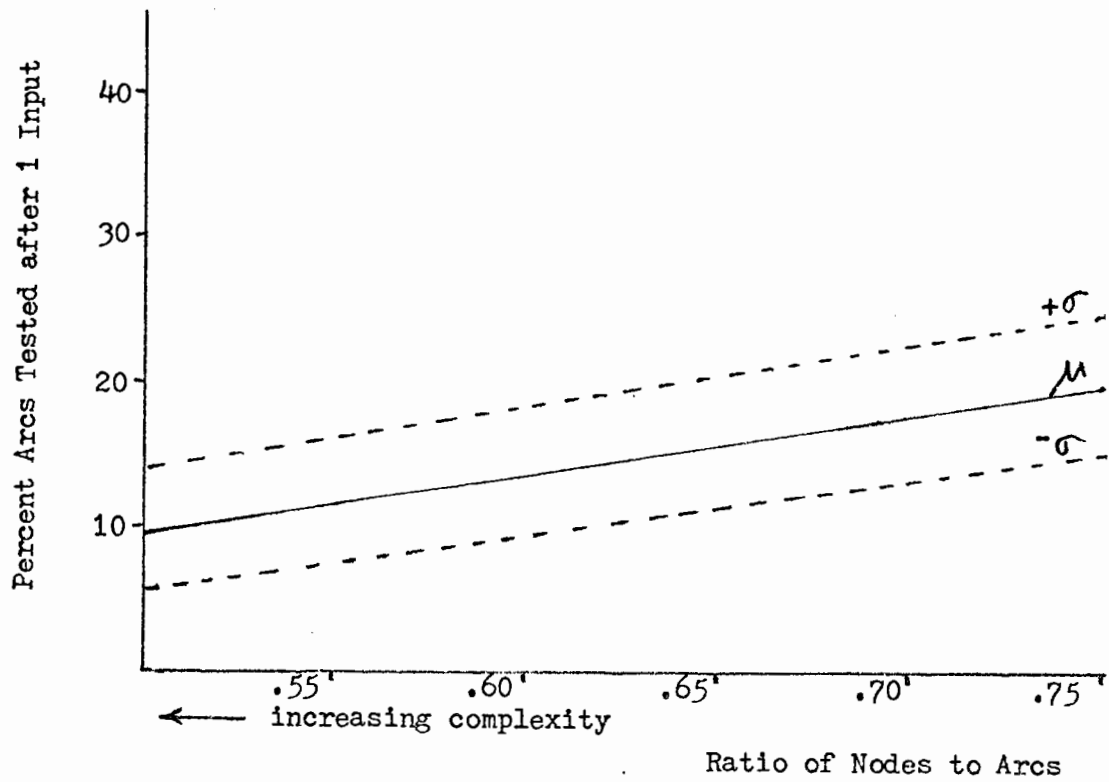


Fig 23 The Effect of LA on Residual Errors

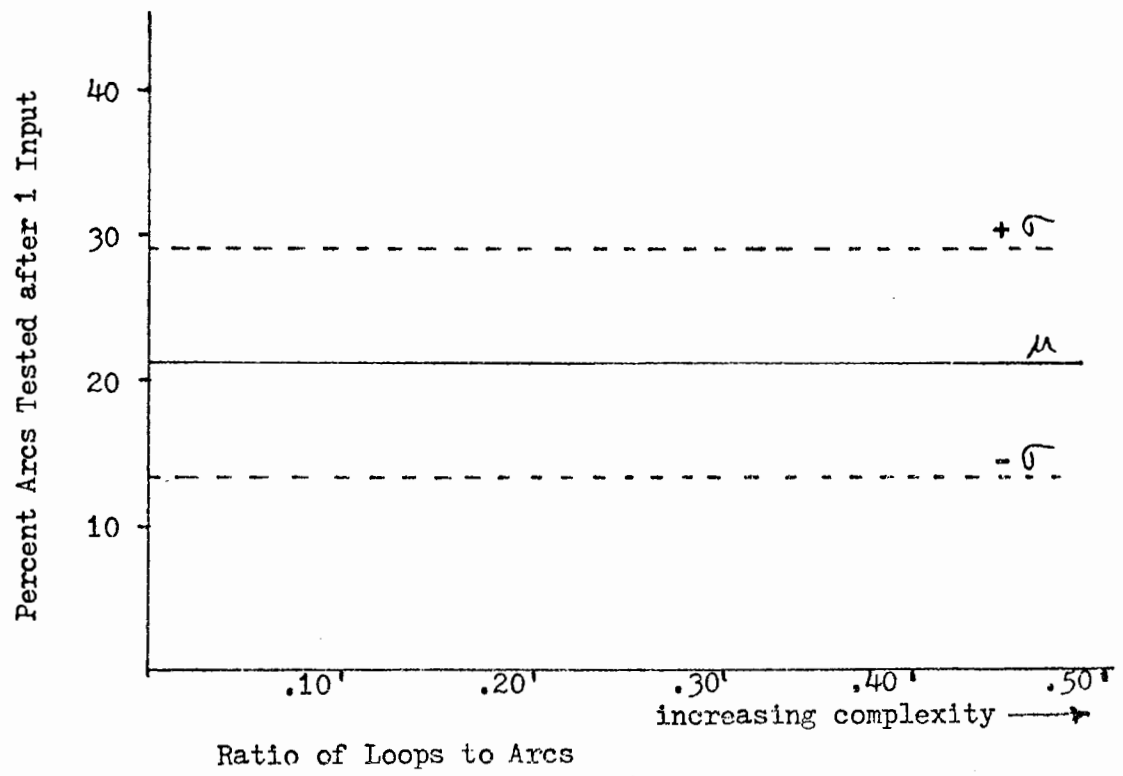


Fig 24 The Effect of LA on Arcs Tested

V. CONCLUSIONS

A. INFERENCES THAT MAY BE DRAWN FROM THE RESULTS

This error simulation model has the potential to be developed into an aid to a manager for the allocation of test effort among the modules of a program. Since it is impossible to test all possible paths in even a moderate sized program, due to the large number of possible paths created as the number of nodes (branch points) increases, it is desirable to know which areas of the program should be tested more thoroughly than others. By writing the program as a directed graph, a manager may use the simulation model to give an estimate of how long to test each module, by using the relationship between error detection and number of inputs.

The simulation model has shown that a structure with many branches is more likely to have a higher percentage of residual errors than a structure with straight line code. In certain instances, a structure with many loops is preferable to one with many branches. The model verified this conclusion in terms of percentage of the arcs tested, where the added loops were an advantage in testing more code, whereas additional branches left many arcs untested. Thus, this model will identify structure types for the manager to select for more thorough testing.

Another feature of the model is the ability to estimate residual errors, percentage of arcs tested, and the amount of repair time required by replicating the testing of a structure with different inputs. By using this type of simulation for the structures which the manager must test, he would obtain useful information for allocating the test effort.

B. FUTURE WORK

The error simulation model should be most useful in the phase of testing computer programs known as individual module development testing. Each module should be tested prior to systems integration testing. With a modification of the definitions presented herein, the model could be also used for systems integration testing, if the modules are defined as the nodes of the directed graph and the inputs to and the outputs from each module are defined as the arcs.

One use of the simulation model would be to write a program in the "normal" fashion and then write a program using structured programming techniques. The two structures could be simulated with the same numbers of initially seeded errors and inputs, and then comparisons of the percent residual errors, percent arcs tested and repair time could be made.

APPENDIX A

FORTTRAN EXAMPLE PROGRAM CODE

Node	Code
	<pre> SUBROUTINE FOR COMPUTING J-N, Y-N, I-N, AND K-N ASYMPTOTIC EXPANSION IS USED FOR LARGE ARGUMENT </pre>
1	<pre> SUBROUTINE BESSEL(X, NORD, BJN, BY, BIN, BK) PI = 3.14152927 GAM = .57721566 </pre>
2	<pre> FN = NORD IF (X - FN - 6.) 1,200,200 XA = X / 2. XB = XA * XA </pre>
	<pre> COMPUTATION OF J-ZERO AND I-ZERO N = 0 </pre>
	<pre> COMPUTATION OF J-N AND I-N BY POWER SERIES </pre>
	<pre> 3 AN = N T = 1. S = -1. </pre>
4	<pre> 10 IF (AN) 9999,20,12 12 T = T * XA / AN AN = AN - 1. </pre>
7,11	<pre> GO TO 10 20 BJN = T BIN = T </pre>
8	<pre> DO 30 K = 1,K DEN = K * (K + N) T = T * XB / DEN </pre>
12	<pre> IF ((BJN + T) - BJN) 25,50,25 25 BJN = BJN + T*S BIN = BIN + T </pre>
15	<pre> 30 S = -S </pre>
14	
17	<pre> 50 IF (N - 1) 75,130,55 </pre>
	<pre> K - N IS COMPUTED FROM ASYMPTOTIC EXPANSION IF X IS GREATER THAN N + 3 </pre>
21	<pre> 55 IF (X - FN - 3.) 1111,1111,200 </pre>
	<pre> CALCULATION OF J-1 AND I-1 </pre>
	<pre> 65 N = 1 BJO = BJN BIO = BIN BYO = BY BKO = BK GO TO 3 </pre>

<u>Node</u>	<u>Code</u>
	C C C COMPUTATION OF K-ZERO AND Y-ZERO
	75 BY = 2./PI * (GAM + LOGF(XA)) * BJN BK = - (GAM + LOGF(XA)) * BIN T = XB S = 1. XI = 1.
22	DO 110 K = 2, K
25	AK = K IF ((BY + T*XI) - BY) 100, 120, 100
	100 BK = BK + T*XI BY = BY + 2./PI*S*T*XI T = T*XB / (AK*AK) XI = XI + 1./AK
28	110 S = -S
27	C C C
29	120 IF (NORD) 9999, 55, 65
	C C C COMPUTATION OF Y-1 AND K-1 BY WRONSKIAN FORMULAS
	130 BY = (BJN*BYO - 2./(PI*X)) / BJO BK = (1. / X - BIN*BKO) / BIO
	C C C Y-N AND K-N BY RECURSION FORMULAS FOR ORDERS HIGHER THAN ONE
20	P = 1. IF (NORD - 1) 9999, 55, 140
	140 BY1 = BY BK1 = BK BY = 2.*P / X*BY1 - BYO BK = 2.*P / X*BY1 + BYO
	C P = P + 1.
23	C IF (NORD - 2) 9999, 150, 146
	146 BYO = BY1 BK = BK1 NORD = NORD - 1 GO TO 140
	C 150 N = P GO TO 3
	C

Node

Code

```
C
C
C      COMPUTATION OF J-N, I-N, K-N, AND Y-N
C      BY ASYMPTOTIC EXPANSIONS
200  C = 4 * NORD * NORD
      D = 8. * X
      CON1 = SQRTF(2./(PI*X))
      CON2 = 1./SQRTF(2.*PI*X)
      CON3 = SQRTF(PI/(2.*X))
      AN = NORD
      PHI = X - (2.*AN + 1.) / 4. * PI
      M = X + 1. + SQRTF(X * X + AN * AN)
      T = (C - 1.) / D
      S = 1.
      U = 1.
      PN = 1.
      QN = T
      BK = 1. + T
      BI = 1. - T
3    C
      DO 240 I = 2, M
      AI = 1
      T = (C - (2.*AI - 1.)**2) / D*T / AI
      BK = BK + T
      BI = BI + T*S
5    IF (S) 220, 9999, 210
      210  PN = PN - T*S*U
          U = -U
          GO TO 230
10   IF ((QN + T) - QN) 230, 241, 230
      220  QN = QN - T*S*U
      230  S = -S
12   240  CONTINUE
13   241  BK = EXPF(-X) * CON3 * BK
      C
      C      ASYMPTOTIC EXPANSION IS USED ONLY FOR K-N
      C      IF X IS BETWEEN N + 3 AND N + 6
      C
16   IF (X - FN - 6.) 1111, 250, 250
      250  BJN = CON1 * (PN * COSF(PHI) - QN * SINF(PHI))
          BY = CON1 * (PN * COSF(PHI) + QN * SINF(PHI))
          BIN = EXPF(X) * CON2 * BI
      C
18,24 1111 NORD = FN
      RETURN
      C
6,9,19, 9999 STOP
26,30
      END
```

APPENDIX B

DIRECTIONS FOR USE OF THE ERROR SIMULATION PROGRAM

Random numbers are drawn using the Naval Postgraduate School Random Number Generator Package LLRANDOM (NPS55LW73061A). These calls are RANDOM for a uniform distribution and EXPON for an exponential distribution.

Printer plots are drawn by the PLOTP routine which is part of the IBM supplied routines. Directed graph plots are drawn by the DRAWP routine available on the Naval Postgraduate School IBM 360/67 System connected to a CALCOMP plotter. The subroutine GRAPHO is limited because it requires the directed graph to be constructed in such a way that the rightmost leg of the tree is the longest leg, which means that the rightmost terminal node is the lowest node.

Histograms are plotted and analyzed by the HISTG routine available on the Naval Postgraduate School IBM 360/67. The functions SIN, COS and ATAN2 are standard IBM FORTRAN-supplied subroutines.

The analysis of loops in the simulation assumed that there were instructions in the arc from i to j and also in the backward arc from j to i. This may not be realistic, but does prove useful when simulating the actual path. When an input reaches a branch point that has a backward arc emanating from it, the model simulates a random number of iterations using the number of instructions in the backward

arc. After completing the simulated looping, it was possible to trace the same path or a different path if there were intermediate branch points. If the same path was selected, there was another chance for the backward arc to be selected. This is a common programming structure, where a loop is executed for so many iterations, a few parameters are changed and the loop is executed again for another or the same number of iterations.

The comment section of the simulation program, titled "Directions for Use", describes the use of each data input to the program as well as the common values utilized during this research. The numbers along the top of the card are the column numbers, the middle row of numbers are sample data and the bottom row of numbers are either the corresponding node for each of the pieces of data above or data item names. The formats used and how each data card was designed are shown below.

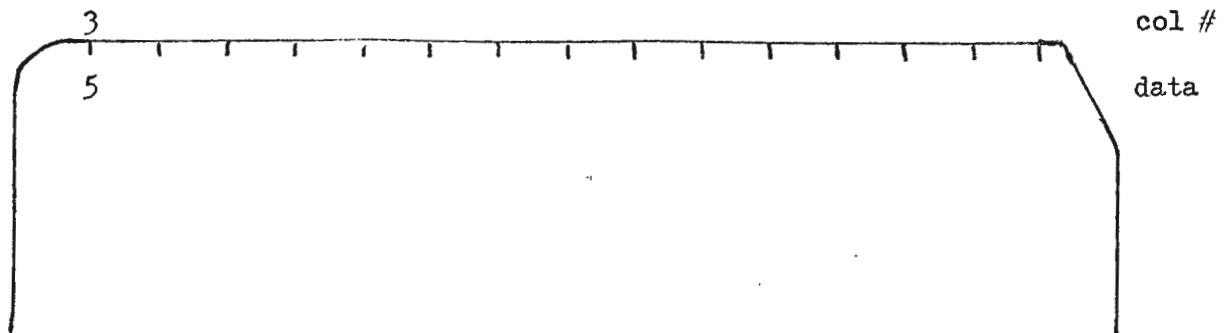


Fig 25 Each item in paragraph A of the directions for use is on a separate card with I3 format except item A(12) which is A5 format (*****).

5	10	15	20	25	30	35	40	45	50	55	60	65	70	75	80	col #
0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	data
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	node #

Fig 26 Adjacency matrix card - 16I5 format.

5	10	15	20	25	30	35	40	45	50	55	60	65	70	75	80	col #
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	data
17	18	19	20	21	22	23	24	25	26	27	28	29	30			node #

Fig 27 Second card of adjacency matrix - based on a graph with 30 nodes.

5	10	15	20	25	30	35	40	45	50	55	60	65	70	75	80	col #
0	0	10.	15.	0	0	0	0	0	0	0	0	0	0	0	0	data
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	node #

Fig 28 Matrix of arc lengths data card - 16F5.0 format.

5	10	15	20	25	30	35	40	45	50	55	60	65	70	75	80	col #
2.	0	0	0	0	0	0	0	0	0	0	0	0	0			data
17	18	19	20	21	22	23	24	25	26	27	28	29	30			node #

Fig 29 Second card of the matrix of arc lengths - based on a graph with 30 nodes.

5	10	col #
22	29	data
N1	N2	id

Fig 30 The two nodes which determine the arc to be added or deleted from the structure - 2I5 format.

10	col #
622150796	data
IZ	id

Fig 31 The seed for the random number generator - format 1I10.

C

C

C

```

206      FORMAT (A5)
207      FORMAT ('0', 12X, 'INPUT ASSUMPTIONS (VARIABLES):'//13X, 'EXECUTIO
N TIME PER INSTRUCTION HAS AN EXPONENTIAL DISTRIBUTION WITH A MEAN
OF', I3, ' MILLISECONDS.'//13X, 'MEAN TIME TO REPAIR AN ERROR HAS
AN EXPONENTIAL DISTRIBUTION WITH A MEAN OF', I3, ' MINUTES.'//13X,
'THE NUMBER OF INPUT PATHS BEING INVESTIGATED IS', I3, '.'//13X,
'ERRORS ARE SEEDDED BY AN EXPONENTIAL DISTRIBUTION WITH A MEAN OF',
I3, ' INSTRUCTIONS PER ERROR.'//13X, 'THE AVERAGE NUMBER OF INSTRU
CTIONS CHANGED WHEN AN ERROR IS FOUND IS', I3, '.')
208      FORMAT ('0', 13X, 'THE NUMBER OF ITERATIONS PER LOOP IS UNIFORMLY
DISTRIBUTED FROM 1 TO', I3, '.')
209      FORMAT ('0', 13X, 'THE MEAN NUMBER OF INSTRUCTIONS BETWEEN EACH NO
DE IS EXPONENTIALLY DISTRIBUTED WITH A MEAN OF', I3, '.')
227      FORMAT ('0', 13X, 'SAMPLE INPUT PATH FROM'//13X, 'NODE TO NODE'//)
230      FORMAT('0', 11X, I4, 4X, I4)
240      FORMAT('0', 13X, 'FOR INPUT NUMBER', I3, ' THE EXECUTION TIME IS',
F8.2, ' SECONDS WITH A TOTAL REPAIR TIME OF', F8.2, ' HOURS'
// 13X, 'AND AN AVERAGE LINK TRAVERSAL TIME OF', F8.2, ' SECONDS')
245      FORMAT ('0', 13X, 'ERROR FOUND'//)
250      FORMAT('1', 13X, I4, ' SEEDDED ERRORS REMAINING'//13X, 'MATRIX OF
SEEDDED ERRORS REMAINING'//7X, 30(I4))
255      FORMAT(' ', 7X, 30(A4))
260      FORMAT('0',      I5, ' *', 30(I4))
265      FORMAT('1', 11X, I3, ' INPUTS'//13X, I4, ' INSTRUCTIONS'//13X,
I3, ' ERRORS SEEDDED'//13X, I3, ' RESIDUAL ERRORS'//13X, F6.2,
' PERCENT RESIDUAL ERRORS'//13X, F6.2, ' PERCENT OF ARCS TESTED'
//)
270      FORMAT('0', 9X, F10.5, ' SECONDS EXECUTION TIME'//12X,
F10.6, ' HOURS TO REPAIR ERRORS'      //12X, F10.6, ' SECONDS AVE
RAGE ARC TRAVERSAL TIME'//)
275      FORMAT('0', 12X, F5.2, ' IS THE RATIO OF ACTUAL TO MAXIMUM NUMBER
OF ARCS'// 13X, F5.2, ' IS THE RATIO OF NODES TO ARCS'//13X,
F5.2, ' IS THE RATIO OF LOOPS TO ARCS'//)
280      FORMAT ('0'//13X, 'HISTOGRAM FOR EXECUTION TIME')
281      FORMAT ('0'//13X, 'HISTOGRAM FOR REPAIR TIME')
282      FORMAT ('0'//13X, 'HISTOGRAM FOR PERCENT RESIDUAL ERRORS')
283      FORMAT ('0'//13X, 'HISTOGRAM FOR PERCENT ARCS TESTED')
284      FORMAT ('1', 1X)
285      FORMAT ('0'//13X, 'EXECUTION TIME VS RESIDUAL ERRORS')
286      FORMAT ('0'//13X, 'EXECUTION TIME VS PERCENT RESIDUAL ERRORS')
287      FORMAT ('0'//13X, 'REPAIR TIME VS RESIDUAL ERRORS')
288      FORMAT ('0'//13X, 'NUMBER OF INPUTS VS RESIDUAL ERRORS')
289      FORMAT ('0'//13X, 'NUMBER OF INSTRUCTIONS VS RESIDUAL ERRORS')
290      FORMAT ('0'//13X, 'RATIO OF ACTUAL TO MAXIMUM NUMBER OF ARCS VS RE
SIDUAL ERRORS')
291      FORMAT ('0'//13X, 'RATIO OF NODES TO ARCS VS RESIDUAL ERRORS')
292      FORMAT ('0'//13X, 'RATIO OF LOOPS TO ARCS VS RESIDUAL ERRORS')

```

1

```

293      FORMAT ('0'//13X,'EXECUTION TIME VS RATIO OF ACTUAL TO MAXIMUM NUM
      BER OF ARCS')
294      FORMAT ('0'//13X,'EXECUTION TIME VS RATIO OF NODES TO ARCS')
295      FORMAT ('0'//13X,'EXECUTION TIME VS RATIO OF LOOPS TO ARCS')
296      FORMAT ('0'//13X,'NUMBER OF INPUTS VS PERCENT OF ARCS TESTED')
297      FORMAT ('0'//13X,'RATIO OF ACTUAL TO MAXIMUM NUMBER OF ARCS VS PE
      RCENT OF ARCS TESTED')
298      FORMAT ('0'//13X,'RATIO OF NODES TO ARCS VS PERCENT OF ARCS TESTE
      D')
299      FORMAT ('0'//13X,'RATIO OF LOOPS TO ARCS VS PERCENT OF ARCS TESTE
      D')
300      FORMAT ('0'//13X,'NUMBER OF INPUTS VS EXECUTION TIME')
301      FORMAT ('0'//13X,'NUMBER OF INPUTS VS PERCENT RESIDUAL ERRORS')
302      FORMAT ('0'//13X,'EXECUTION TIME VS NUMBER OF INSTRUCTIONS')
303      FORMAT ('1',59X,'DATA SUMMARY'//62X,'PERCENT',3X,'PERCENT',
      2X,'EXECUTION',3X,'REPAIR',4X,'ACTUAL',4X,'NODES',5X,
      'LOOPS',4X,'RUN',35X,'ERRORS',3X,'RESIDUAL',2X,'RESIDUAL',
      4X,'ARCS',6X,'TIME',6X,'TIME',5X,'TO MAX',6X,'TO',8X,
      'TO',2X,
      'NUMBER',5X,'NODES',4X,'INPUTS',4X,'INSTR.',4X,'SEEDED',
      4X,'ERRORS',4X,'ERRORS',4X,'TESTED',4X,'(SEC)',7X,
      '(HRS)',4X,'ARCS',6X,'ARCS',6X,'ARCS')
304      FORMAT ('0',2X,I3,1X,'*',5X,I3,7X,F5.0,3(5X,F5.0),
      2(4X,F6.2),2X,2F10.5,3X,F5.3,2(5X,F5.3))
305      FORMAT ('0',13X,'THE LAST SEED USED WAS',I11,'.')
      READ IN THE TYPE OF SIMULATION TO BE RUN (SIMNUM)

      ***READ (5,100) SIMNUM
      READ IN NUMBER OF INPUTS TO BE USED (MINPUT)

      ***READ (5,100) MINPUT
      READ IN THE MEAN NUMBER OF INSTRUCTIONS BETWEEN EACH NODE (MEANLN)

      ***READ (5,100) MEANLN
      READ IN THE NUMBER OF GRAPHS TO BE EVALUATED (NUMOUT)

      ***READ (5,100) NUMOUT
      READ IN WHETHER OR NOT THE NUMBER OF INSTRUCTIONS IS TO BE READ IN

      ***READ (5,100) INREAD
      READ IN WHETHER DELETING AN ARC FROM THE STRUCTURE OR

      ADDING AN ARC TO THE STRUCTURE.

      ***READ (5,100) DELADD
      READ IN THE MEAN TIME TO EXECUTE AN INSTRUCTION IN MILLISECONDS

```

(CONTINUED ON PAGE 4)

```
***READ (5,100) ITIME
```

***READ (5,100) MMTTR

```
***READ (5,100) MEANER
```

```
***READ (5,100) MEANIT
```

***READ (5,100) AVCHNG

***READ (5,206) ASTER

```
***READ (5,100) N
```

```

      +-----+
      | DO      |
      | 20      |
      | I = 1, N |
      +-----+

```

```
*      *      *
```

```
*      IF      *
```

```
* SIMNUM.EQ.1 *
```

```
*      *
```

```
*      *
```

```
T      | WRITE(6,151)
```

66

```
* . * IF * .  
* SIMNUM.EQ.2 *  
* . * T | WRITE(6,152)  
* F  
* . * IF * .  
* SIMNUM.EQ.3 *  
* . * T | WRITE(6,153)  
* F  
* . * IF * .  
* SIMNUM.EQ.4 *  
* . * T | WRITE(6,154)  
* F  
* . * IF * .  
* SIMNUM.EQ.5 *  
* . * T | WRITE(6,155)  
* F  
* . * IF * .  
* INREAD.EQ.1 *  
* . * T | WRITE(6,156)  
* F
```

```

***WRITE (6,207) ITIME, MMTR, MINPUT, MEANER, AVCHNG
***WRITE (6,208) MEANIT
(CONTINUED ON PAGE 6)

```

```
***WRITE (6,204) NUMOUT
```

```

NNOUT=1
IX    =71286223
IW    =IX

```

```
| MAXARC =N*(N-1)
```

```

+-----+
+ DO      +
+ 57      +
+ I = 1,N +
+-----+

```

```
1 NUMPTS(I) = I
```

```
1 ASTOUT(I) = ASTER
```

CALL DIGRAF

59 + 00 +
+++++ 62 +
+ + I = 1, N +

```

+-----+
+ DO +
+-----+
+ J = 1,N +
+-----+

```

```
1 ISEED(I,J) = 0
```

62 +++++CONTINUE

(CONTINUED ON PAGE 7)

```
*      *      *      *      *  
*      *      IF      *      *  
*      SIMNUM.EQ.1    *      *  
*      *      *      *      *  
*      *      T      | IX=1722632  
*      *      *      *      *  
*      F      |  
*      *      *
```

THIS SEED CAN BE CHANGED TO THE LAST VALUE OF THE SEED ON THE PREVIOUS TEST IF A CONTINUATION OF THE DATA IS DESIRED.

```

      * . * . * .
    * IF
    * SIMNUM.EQ.5 *
    * . * . * .
      * . * . * .
        F
          T | READ(5,108)IX
            -----

```

```
| ARCS = 0.  
| INST = 0  
| NSEED = 0
```

```

+-----+
+ DO      +
+ 65      +
+ I=1,N   +
+-----+
+
+-----+
+ DO      +
+ 65      +
+ J=1,N   +
+-----+

```

```

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040

```

69

CALCULATE THE NUMBER OF ARCS IN THE PROGRAM

```
| ARCS = ARCS + 1.
```

CALCULATE THE NUMBER OF INSTRUCTIONS

```
1 INST = INST+X(I,J)
```

CALCULATE THE NUMBER OF ERRORS SEEDED

```
| NSEED= NSEED+ ISEED(I,J)
```

65 +++++CONTINUE

CALCULATE THE NUMBER OF LOOPS IN THE PROGRAM

```
|  LOOPS=0
```

```

+-----+
+ DO                                     +
+      68                               +
+ I = 1, N                             +
+-----+
+
+-----+
+ DO                                     +
+      68                               +
+ J = 1, I                             +
+-----+

```

(CONTINUED ON PAGE 9)

(CONTINUED ON PAGE 12)

- - - - -		- - - - -
	78	
- - - - -		- - - - -

95

```

      |
-----
| NODE = L
-----
      |
      |-----|
      | 79  |
      |-----|

```

AVLINK IS THE AVERAGE ARC TRAVERSAL TIME IN MINUTES

96

$$\text{AVLINK} = (\text{TIME} / \text{ARCS})$$

```
***WRITE (6,240) INPUT, TIME, TOTREP, AVLINK
```

```
*      *      *      *      *
*      *      IF      *      *
*      *      SIMNUM.EQ.2    *
*      *      *      *      *
*      *      *      *      *
```

T | 97

```
TTIME(NNOUT) = TTIME(NNOUT) + TIME
TREP(NNOUT) = TREP(NNOUT) + TOTREP
TLINK(NNOUT) = TLINK(NNOUT) + AVLINK
```

199

97

! NOUT = NNOUT-1

(CONTINUED ON PAGE 14)

```
TLINK(1) = AVLINK
TREP(1) = TOTREP
TTIME(1) = TIME
```

99

```

| TLINK(NNOUT) = TLINK(NCUT) + AVLINK
| TREP(NNOUT) = TREP(NOUT) + TOTREP
| TTIME(NNOUT) = TTIME(NOUT) + TIME

```

```
TIME = 0.  
TOTREP = 0.  
AVLINK = 0.  
INPUT = INPUT + 1
```

```
| KOUNT=0
```

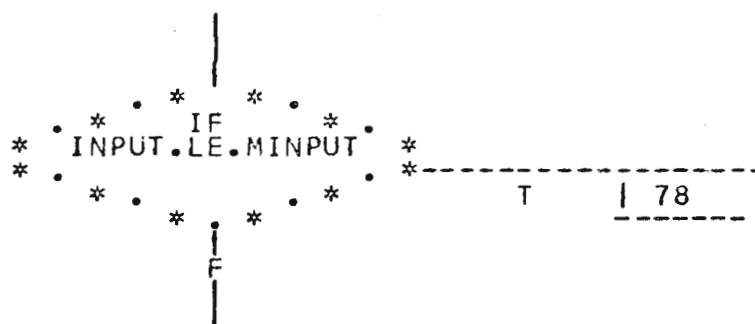
```

+-----+
+ DO                                         +
+ 101                                       +
+ I = 1,N                                   +
+-----+
+
+-----+
+ DO                                         +
+ 101                                       +
+ J = 1,N                                   +
+-----+

```

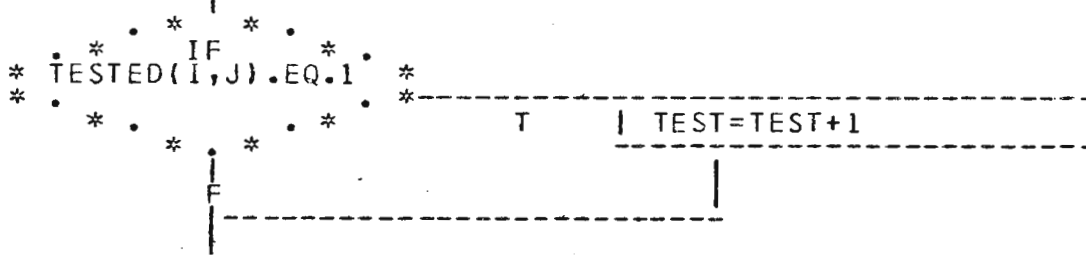
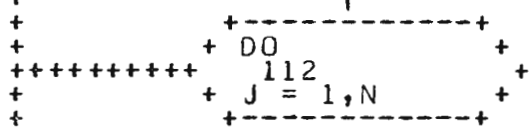
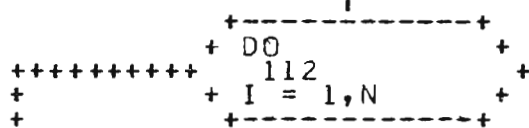
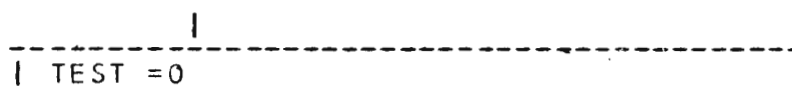
(CONTINUED ON PAGE 15)

(CONTINUED ON PAGE 16)

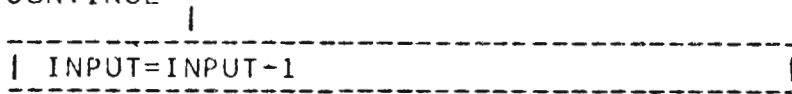


CALCULATE THE NUMBER OF ARCS TESTED

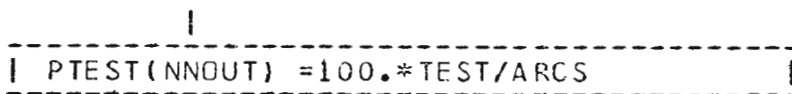
111



112 ++++++CONTINUE



PTEST IS THE PERCENT OF ARCS TESTED



F IS THE RATIO OF ACTUAL TO MAXIMUM NUMBER OF ARCS



(CONTINUED ON PAGE 17)

```
1 FNA(NNOUT) = N/ARCS
```

```

FLA(NNOUT) = LOOPS/ARCS
TINST(NNOUT) = INST
TINPUT(NNOUT) = INPUT
TSEED(NNOUT) = NSEED
TKOUNT(NNOUT) = KOUNT
PSEED(NNOUT) = 100.*(TKOUNT(NNOUT)
/ NSEED)

```

```
***WRITE (6,275) F(NNOUT), FNA(NNOUT), FLA(NNOUT)
```

```

NNOUT=NNOUT+1

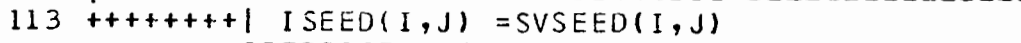
```

```
*      *      *      *      *      *      *      *      *      *
*      *      IF      *      *      *      *      *      *      *
*      NNOUT.GT.NUMOUT      *      *      *      *      *      *
*      *      *      *      *      *      *      *      *      *
*      *      *      *      *      *      *      *      *      *
```

T | 118

```
*      *      *      *      *
*      *      IF      *      *
*      *      SIMNUM.EQ.1    *      *
*      *      *      *      *      *
*      *      *      *      *      *      *      *
*      *      *      T      |      MEANLN=MEANLN+1
```

(CONTINUED ON PAGE 18)



(CONTINUED ON PAGE 19)

```

      *
    *   IF
  * SIMNUM.EQ.5
*-----T | 63 |

```

115

```
*      *      *      *      *
*      *      IF      *      *
*      DELADD.EQ.1    *      *
*      *      *      *      *
*      *      *      *      *
*      *      *      *      *
*      *      *      *      *
*      *      *      *      *
```

```
116      ***WRITE (6,180) (NUMPTS(I), I=1,N)
      ***WRITE (6,255) (ASTOUT(I), I = 1,N)
```

```

      +-----+
      | DO      |
      | 117     |
      | I = 1,N |
      +-----+

```

```

117  +++++**WRITE (6,260)  NUMPTS(I), (NODES(I,J), J=1,N)
      ***WRITE (6,203)
      |
      +-----+
      | 63   |

```

```
118      ***WRITE (6,305) IZ
      ***WRITE (6,284)
      CALL PLOTP (TTIME, TKOUNT, NUMOUT, 0)
      ***WRITE (6,285)
      ***WRITE (6,284)
      (CONTINUED ON PAGE 20)
```

IF
SIMNUM.NE.2

T 119

F

119

```
* . * . * IF * . *  
*(SIMNUM.EQ.2).OR.(SIMNUM.EQ.5)  
* . * . * . * ----- T | 121 |  
      * . *  
      F
```

(CONTINUED ON PAGE 21)

```
*      *      *      *      *  
*,    IF  
SIMNUM.EQ.1  
*      *      *      *      *
```

T | 121

83

```

      * . * *
      * * IF *
* (SIMNUM.NE.3).AND.(SIMNUM.NE.4)
* * * * *
      * . * *
      * * T | CALLGRAPHO

```

IF
SIMNUM.NE.5

T | 126

126 CONTINUE
(CONTINUED ON PAGE 23)

SUBROUTINE DIGRAF

CC

C

CONSTRUCT THE DIRECTED GRAPH

C

C

CC

COMMON/ALLL/N, NODES(30,30)

COMMON/ARRAY/X(30,30), NUMPTS(30), ASTOUT(30), AVCHNG

COMMON/GEN/MEANLN, IX

COMMON/ERROR/ISEED(30,30), ITER(30,30), MEANER, INST, MEANIT, IZ

COMMON/GRAF/INREAD

INTEGER*4 CHANGE, AVCHNG

100 FORMAT (16F5.0)

130 FORMAT ('0', 10X, 14, 1X, 'NODES IN DIRECTED GRAPH.'//)

140 FORMAT('0', 13X, 'MATRIX OF ARC LENGTHS'//7X, 30(I4))

150 FORMAT ('0', 15, ' *', 30(F4.0))

160 FORMAT ('0', 13X, 'NOTE: 0.0 INDICATES NO ARC FROM I TO J'//)

180 FORMAT ('0', 13X, 'ADJACENCY MATRIX'//7X, 30(I4))

190 FORMAT ('0', 15, ' *', 30(I4))

200 FORMAT ('0', 13X, 'NOTE: NONEXISTENCE OF AN ARC IS INDICATED BY A ZERO'//)

210 FORMAT (' ', 7X, 30(A4))

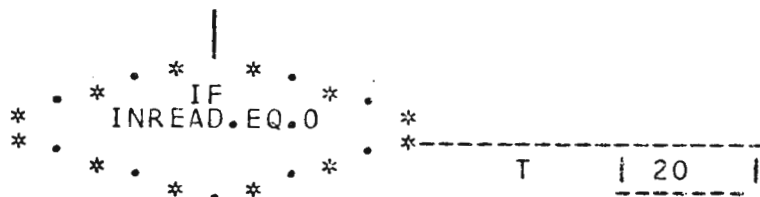
222 FORMAT (' ', 7X, 30(A4))

225 FORMAT ('0', 15, ' *', 30(I4))

230 FORMAT ('0', 13X, 'LOOP ITERATIONS'//7X, 30(I4))

IF INREAD = 0 RANDOMLY GENERATE INSTRUCTION LENGTHS

IF INREAD = 1 READ IN INSTRUCTION LENGTHS



(CONTINUED ON PAGE 2)


```
+  
+  
+  
+  
+  
+  
+  
+  
+  
+  
+  
+  
+  
+  
+  
+  
+  
  
      *   *   *  
    . IF  
  * J.GT.I *  
  *       *  
    .     .  
      *   *  
        F  
          |  
CALL RANDOM (IX, TITER, 1)  
          |  
-----| ITER(I,J) =1+MEANIT*TITER |-----  
          |  
55 ++++++++CONTINUE  
    CREATE LABELS FOR THE OUTPUT ARRAY  
  
    ***WRITE (6,230) (NUMPTS(I), I=1,N)  
    ***WRITE (6,222) (ASTOUT(I), I=1,N)  
          |  
      +-----+  
      + DO  
++++++73  
      + I = 1,N  
      +-----+  
          |  
73 +++++***WRITE (6,225) NUMPTS(I), (ITER(I,J), J=1,N)  
    RETURN
```

SUBROUTINE ADDARC

CC

C

ADDARC ADDS AN ARC BETWEEN TWO NODES

C

C

CC

COMMON/ALLL/N, NODES(30,30)

COMMON/ARRAY/X(30,30), NUMPTS(30), ASTOUT(30), AVCHNG

COMMON/GEN/MEANLN, IX

100

FORMAT (2I5)

110

FORMAT ('1', 13X, 'A NEW ARC HAS BEEN ADDED FROM NODE', I3,
' TO NODE', I3, ' OF LENGTH', F4.0)

ADD A NEW ARC FROM N1 TO N2

***READ (5,100) N1, N2

NODES(N1,N2) =1

CALL EXPON (IX, XLNTH, 1)

MEANLN IS THE MEAN LENGTH OF THE ARC TO BE EXPONENTIALLY DISTRIB.

X(N1,N2) =MEANLN*XLNTH+1.0

***WRITE (6,110) N1, N2, X(N1,N2)

RETURN

END

SUBROUTINE DELARC

CC

C

DELARC DELETES AN ARC BETWEEN TWO NODES

C

C

CC

COMMON/ALLL/N, NODES(30,30)

COMMON/ARRAY/X(30,30), NUMPTS(30), ASTOUT(30), AVCHNG

COMMON/ERROR/ISEED(30,30), ITER(30,30), MEANER, INST, MEANIT, IZ

COMMON/OUT/NSEED, INPUT, SVSEED(30,30)

INTEGER SVSEED

100 FORMAT (2I5)

110 FORMAT ('1', 13X, 'AN ARC WAS DELETED FROM NODE', I3,
' TO NODE', I3, '.')

***READ (5,100) N1, N2

NODES(N1,N2) = 0 X(N1,N2) = 0.0 ISEED(N1,N2) = 0
--

SVSEED(N1, N2) = 0

***WRITE (6,110) N1, N2

RETURN

END

SUBROUTINE SEED

[illegible]

SEED THE GRAPH WITH ERRORS

CCCCCCCCCCCCCCCCCCCCCCCCCC

COMMON/ALL/N, NODES(30,30)

COMMON/ARRAY/X(30,30), NUMPTS(30), ASTOUT(30), AVCHNG

COMMON/ERROR/ISEED(30,30), ITER(30,30), MEANER, INST, MEANIT, IZ

COMMON/OUT/NSEED, INPUT, SVSEED(30,30)

COMMON/GEN/MEANLN, IX

INTEGER*4 CHANGE, AVCHNG, SVSEED

```

DIMENSION ER(30), IER(30)

```

```
205      FORMAT ('0', 13X, 'SEED ERRORS AT INSTRUCTION INTERVALS'// 10X,
              30I4)
```

```
210      FORMAT ('1'// 13X, 'TOTAL NUMBER OF INSTRUCTIONS IS', I5)
```

```
220      FORMAT ('0'// 13X, 'TOTAL NUMBER OF ERRORS SEEDS IS', I4// 13X,  
              'THE ERROR MATRIX'//7X, 30(I4))
```

222 FORMAT (' ', 7X, 30(A4))

```
225      FORMAT ('0',      I5, ' *', 30(I4))
```

GENERATE ERRORS WITH EXPONENTIAL DISTRIBUTION

CALL EXPON(IX, ER, N)

SPREAD ERRORS TO GIVE MEAN OF MEANER

```

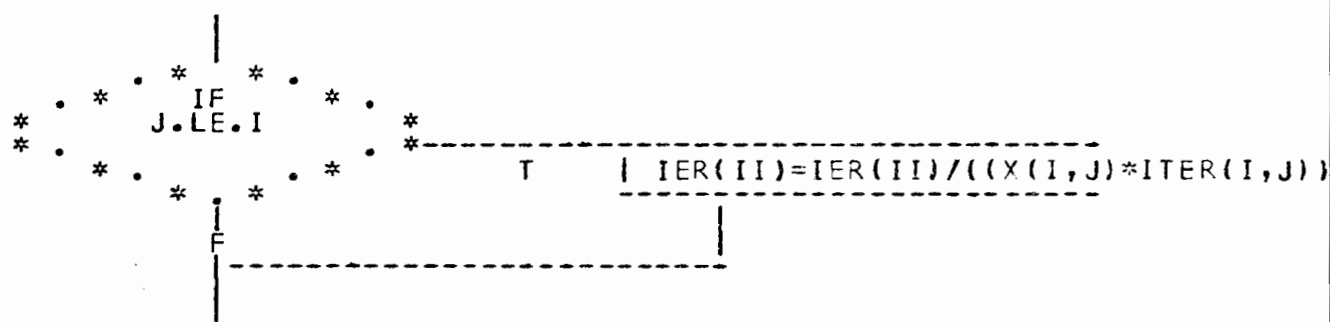
+-----+
+ DO 60 +
+ I = 1, N +
+-----+
|
+-----+
+ IER(I) = MEANER*ER(I) +
+-----+
|

```

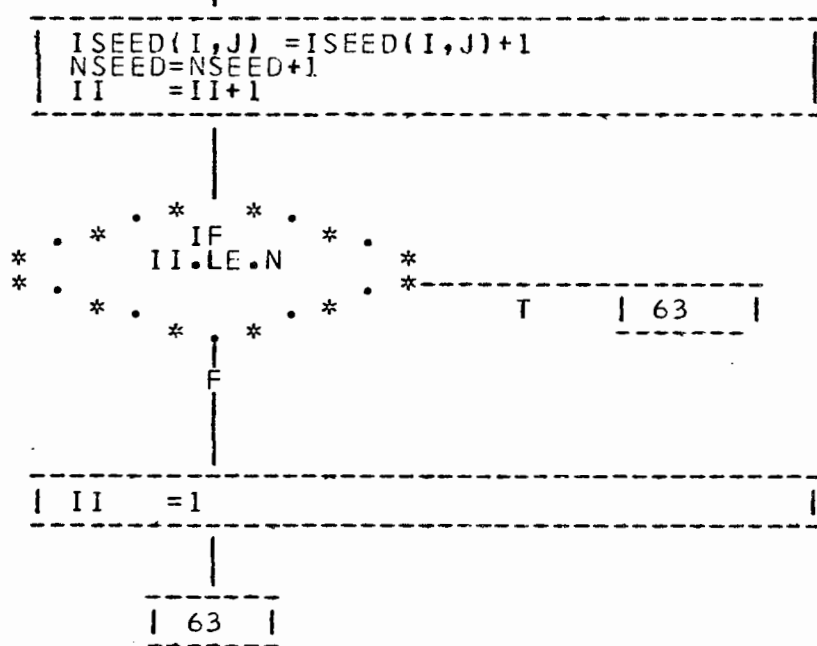
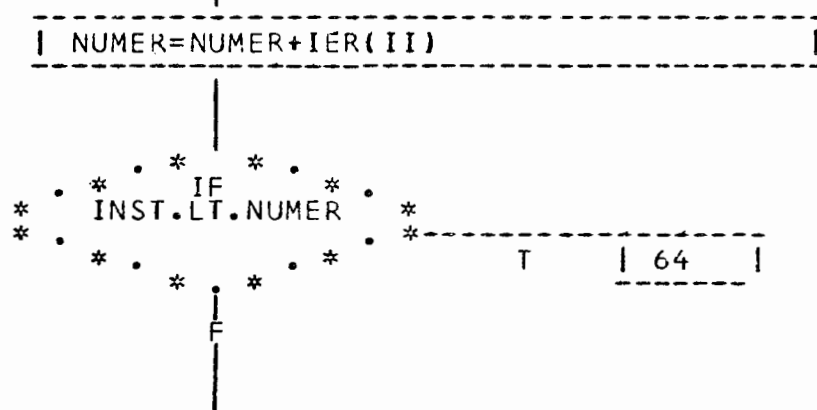
92

(CONTINUED ON PAGE 2)

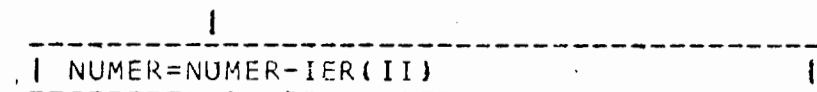
THROUGH THE LOOP



63



64



(CONTINUED ON PAGE 4)

```

+
+
+
65 ++++++CONTINUE
+
70 ++++++CONTINUE
+
+-----+
+ DO
+ 71
+ I = 1,N
+-----+
+
+-----+
+ DO
+ 71
+ J = 1,N
+-----+
+
+
SVSEED IS THE SAVE COPY OF THE ORIGINAL SEEDED ERRORS
+
+-----+
71 ++++++| SVSEED(I,J) =ISEED(I,J) |
+-----+
+
+
NSEED IS THE NUMBER OF ERRORS SEEDED

***WRITE (6,210) INST
***WRITE (6,205) (IER(I), I = 1,N)
***WRITE (6,220) NSEED, (NUMPTS(I), I = 1,N)
***WRITE (6,222) (ASTOUT(I), I=1,N)
+
+-----+
+ DO
+ 75
+ I = 1,N
+-----+
+
+
***WRITE (6,225) NUMPTS(I), (ISEED(I,J), J=1,N)
75 ++++++CONTINUE

```

RETURN

END

SUBROUTINE NUSEED

CC

C

CORRECTING PREVIOUS ERROR MAY CREATE A NEW ERROR

C

C

CC

COMMON/ALL/N, NODES(30,30)

COMMON/ARRAY/X(30,30), NUMPTS(30), ASTOUT(30), AVCHNG

COMMON/ERROR/ISEED(30,30), ITER(30,30), MEANER, INST, MEANIT, IZ

COMMON/OUT/NSEED, INPUT, SVSEED(30,30)

COMMON/NEW/IW

INTEGER*4 CHANGE, AVCHNG

DIMENSION ERR(2)

247

FORMAT ('0', 13X, 'NEW ERROR INSERTED FROM NODE', I4, ' TO NODE',
I4//)

NUM = 0

THE ERROR IS EQUALLY LIKELY TO RESULT IN CHANGES TO ALL ARCS

CALL RANDOM (IW, ERR, 2)

I = 1 + N * ERR(1)
J = 1 + N * ERR(2)

NEW ERROR CREATED AT (I,J)

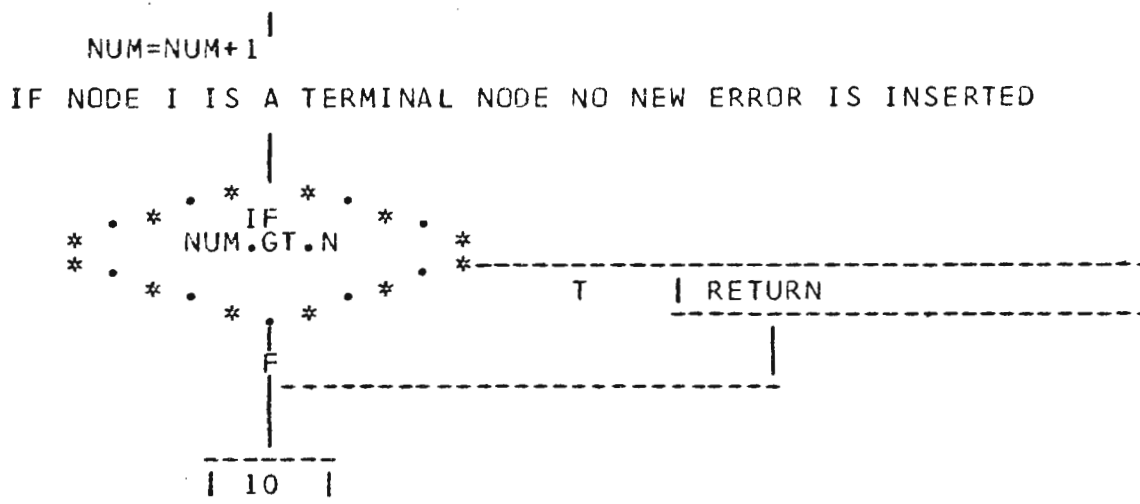
IS THIS A NODE OF THE DIRECTED GRAPH

10

IF (NODES(I,J).EQ.1) THEN
T 20

J = J + 1
IF (J.GT.N) J = 1

(CONTINUED ON PAGE 2)



CHANCE OF INSERTING A NEW ERROR IS BASED ON THE NUMBER OF INSTRUCTIONS THAT MUST BE CHANGED

20 CALL RANDOM (IW, ERNU, 1)
 AVCHNG IS THE ASSUMED (INPUT) AVERAGE NUMBER OF INSTRUCTIONS CHANGED WHEN AN ERROR IS FOUND
 CHANGE IS THE UNIFORMLY DISTRIBUTED NUMBER OF INSTRUCTIONS CHANGED

$$| \text{CHANGE} = 1 + \text{AVCHNG} * \text{ERNU} * 2. |$$

CRATIO IS THE RATIO OF INSTRUCTIONS CHANGED TO THE NUMBER OF INSTRUCTIONS IN THE UNIFORMLY RANDOMLY CHOSEN ARC

$$| \text{CRATIO} = \text{CHANGE} / \text{X(I,J)} |$$

THE CRITERIA FOR INSERTING A NEW ERROR IS UNIFORMLY DISTRIBUTED

CALL RANDOM (IW, RATIO, 1)
 IF THE CHANGE RATIO (CRATIO) IS GREATER THAN THE CRITERIA (RATIO),
 THEN A NEW ERROR IS INSERTED

(CONTINUED ON PAGE 3)

SUBROUTINE GRAPHO

CC

C

GRAPHICALLY DISPLAY THE DIRECTED GRAPH USING THE CALCOMP PLOTTER C

C

CC

COMMON/ALLL/N, NODES(30,30)

DIMENSION ND(30,30), X(30), XX(2), Y(30), YY(2)

DIMENSION XL(36), YL(36), THETA(36)

INTEGER*4 ITB(12)/12*0/

REAL*4 RTB(28)/28*0.0/

ITB IS DRAWP ARRAY

ITB(1)	=1
ITB(2)	=0
ITB(3)	=9
ITB(4)	=15
ITB(8)	=2
ITB(9)	=0
ITB(10)	=2
ITB(11)	=0

RTB IS DRAWP ARRAY

RTB(1)	=1.0
RTB(2)	=1.0

```

+-----+
+ DO      +
+ 4      +
+ I = 1,N +
+-----+
+
+ DO      +
+ 4      +
+ J = 1,N +
+-----+

```

ND IS ARRAY USED TO CHECK FOR PREDECESSORS

ND(I,J)	=0
---------	----

(CONTINUED ON PAGE 2)

```

+
+
+
4 ++++++CONTINUE

```

```

| X(1) =4.5
| Y(1) =15.0
| X(2) =X(1)
|

```

CONST IS Y-DIRECTION SCALE FACTOR

```

| CONST=30./N
| Y(2) =Y(1)-CONST
|

```

CALL DRAWP(2,X,Y,ITB,RTB)

```

| XCONST =1.5
|

```

KOUNT IS X-DIRECTION SHRINK FACTOR - SHRINKS AFTER EVERY OTHER
FANOUT

```

| KOUNT=1
|

```

LEVEL IS THE LEVEL IN THE DIRECTED GRAPH

```

| LEVEL=2
| NN =N-1
| ITB(1) =2
|

```

LABEL IS THE NUMBER OF EACH NODE

```

| LABEL=2
|

```

```

+-----+
+ DO 5 +
+++++++ + M = 1,N +
+-----+
+
+
+

```

(CONTINUED ON PAGE 3)

```

      *   *   *
    *   *   IF   *   *
*   NODES(1,M).EQ.0   *
*   *   *   *   *   *-----T | 5 |
      *   *   *

```

KK IS THE LAST NODE IN THE GIVEN LEVEL

$$\begin{array}{lcl} \text{KK} & = & M \\ \text{KT} & = & \text{KK} + 1 \end{array}$$

```

      *   *   *
      *   IF   *
* NODES(1,KT).EQ.0 *
*   *   *   *   *
                        T | 6 |

```

```
5  ++++++CONTINUE
```

KM IS THE LAST NODE IN THE NEXT LEVEL

```

6      +-----+
+++++++ DO          +
+               7   +
+       M = KK,N    +
+     +-----+

```

```

      *   *   *
    . *   IF   * .
* NODES(KK,M).EQ.0 *
*   *       *
                                T | 7 |

```

$$\begin{aligned} KM &= M \\ KX &= KM+1 \end{aligned}$$

(CONTINUED ON PAGE 4)


```
*      *      *      *      *
*      *      IF      *      *
*      NUMSUC.NE.1    *      *
*      *      *      *      *
*      *      *      *      *
```

T | 30 |

```

L      = LABEL+1
LABEL = LABEL+1
ND(I,L) = 1
X(L)   = X(I)
Y(L)   = Y(I) - CONST
XX(1)  = X(I)
XX(2)  = X(L)
YY(1)  = Y(I)
YY(2)  = Y(L)

```

IS THERE A NODE ABOVE THAT NEEDS CONNECTING TO THIS NODE

$$+ \text{DO} \quad +$$

$$+ \quad 28 \quad +$$

$$+ \text{K} = 1,11 \quad +$$

$$+ \text{-----} +$$

```
*      *      *      *      *
*      *      IF      *      *
*      ND(K,L).EQ.1      *      *
*      *      *      *      *
*      *      *      T      |      28      |
```

```
*      *      *      *      *
*      *      IF      *      *
*      *      NODES(K,L).EQ.0      *
*      *      *      *      *
*      *      *      *      *      T      |      28      |
```

$$\begin{cases} XX(1) = X(K) \\ YY(1) = Y(K) \end{cases}$$

(CONTINUED ON PAGE 7)

[illegible]

(CONTINUED ON PAGE 8)

$$YY(1) = Y(I)$$
$$\begin{aligned} XX(2) &= X(M) \\ YY(2) &= Y(M) \end{aligned}$$

CALL DRAWP(2,XX,YY,ITB,RTB)

IS THERE A NODE ABOVE THAT NEEDS CONNECTING TO THIS NODE

DO

K=1.11

```

*      IF
ND(K,M).EQ.1

```

T 1 34

三

```

      *      IF      *
      NODES(K,M).EQ.0

```

T 1 34

三

```

XX(1)=X(K)
YY(1)=Y(K)
XX(2)=X(M)
YY(2)=Y(M)
ND(K,M)=1

```

CALL DRAWP(2,XX,YY,ITB,RTB)

34 +++++CONTINUE

35 +++++CONTINUE

```

LABEL=LABEL+NUMSUC
KOUNT=KOUNT+1

```

(CONTINUED ON PAGE 9)

```
*      *      *      *
*      *      IF      *
*      KOUNT.LT.2     *
*      *              *
*      *              *
*      *              *
*      *              *
*      *              *
*      *              *
*      *              *
*      *              *
*      *              *
*      *              *
```

T | 44

F

```
XCONST=XCONST/1.5  
KOUNT=0
```

* . * * *
* . * I F * .
* . I . L T . K K *
* . * . *
* . * . *
* . * . *
* . * . *
* . * . *
* . * . *

T | 50

F

```
| LEVEL=LEVEL+1
```

```

+-----+
+ DO      +
+ 45      +
+ M = I, N +
+-----+

```

```
*      *      IF      *
*      *      NODES(I,M).EQ.0      *
```

```
*      *      T      |      45      |
```

```
*      *      F
```

```
| KK      =M
```

107

```
+  
+ KT=KK+1  
+ |  
+ * . * IF * . *  
+ * NODES( I ,KT) .EQ.0 *  
+ * . * T | 46 |  
+ * F  
+ |  
  
45 +++++CONTINUE  
+ DETERMINE THE LAST NODE IN THE NEXT LEVEL AFTER THE PRESENT LEVEL  
+ |  
+ +- -+- DO +- -+  
+ | M = KK,N |  
+ +- -+-  
+ |  
+ * . * IF * . *  
+ * NODES(KK,M) .EQ.0 *  
+ * . * T | 47 |  
+ * F  
+ |  
+ +- -+- KM =M +- -+  
+ | KX =KM+1 |  
+ +- -+-  
+ |  
+ * . * IF * . *  
+ * NODES(KK,KX) .EQ.0 *  
+ * . * T | 50 |  
+ * F  
+ |  
  
47 +++++CONTINUE  
50 +++++CONTINUE
```

GRAPH LOOPS

|
(CONTINUED ON PAGE 11)

```
+-----+  
| DO  
90  
I = 2,N  
+-----+  
  
+-----+  
| K = I-1  
+-----+  
  
+-----+  
| DO  
80  
J = 1,K  
+-----+  
  
+-----+  
IF  
NODES(I,J).EQ.0  
+-----+ T | 80 |  
F  
  
+-----+  
PI = 3.1415926  
+-----+  
  
ESTABLISH THE DIRECTION OF CURVE FOR LOOP  
  
+-----+  
IF  
X(I).LT.X(J)  
+-----+ T | PI=-PI |  
F  
  
STARTING POINT FOR CURVE  
  
+-----+  
XL(1)=X(I)  
YL(1)=Y(I)  
+-----+  
  
ENDING POINT FOR THE CURVE  
  
+-----+  
XL(36) = X(J)  
YL(36) = Y(J)  
+-----+
```

(CONTINUED ON PAGE 12)

XDIF AND YDIF ARE THE DIFFERENCE BETWEEN THE X AND Y VALUES RESP.

```

| XDIF =X(I)-X(J)
| YDIF =Y(I)-Y(J)
|

```

R IS THE RADIUS OF THE LOOP

```

| R      =0.5*SQRT(XDIF**2+YDIF**2)
|

```

(XC,YC) IS THE CENTER OF THE LOOP

```

| XC      =(X(I)+X(J))/2.
| YC      =(Y(I)+Y(J))/2.
|

```

THETE(1) IS THE INITIAL ANGLE

```

| THETA(1) =ATAN2(YDIF,XDIF)
|

```

CREATE 36 POINTS IN A SEMICIRCLE FORMING THE LOOP

```

+-----+
+ DO          +
+ 70          +
+ L = 2,35    +
+-----+

```

```

| M      =L-1
| THETA(L) =THETA(M)+PI/36.
| XL(L)=R*COS(THETA(L))+XC
| YL(L)=R*SIN(THETA(L))+YC
|

```

70 ++++++CONTINUE

CALL DRAWP(36,XL,YL,ITB,RTB)

80 ++++++CONTINUE

90 ++++++CONTINUE

DRAW DIAMONDS AT EACH NODE

```

| ITB(1) =3
| ITB(2) =4
|

```

CALL DRAWP(N,X,Y,ITB,RTB)

RETURN

END

(This page intentionally blank)

112

CC

CCCCCCCCCCCCCCCCCCCCCCCCCC

C 1. TO RUN THE SIMULATION WHICH VARIES THE NUMBER OF INSTRUCTIONS

- C A. NORMAL DATA INCLUDES:

- C. INPUT ADJACENCY MATRIX

- C A. NORMAL DATA INCLUDES:

- B. INPUT ADJACENCY MATRIX
 C. IF INREAD = 1 INPUT MATRIX OF ARC LENGTHS.
3. TO RUN THE SIMULATION WHICH VARIES THE NUMBER OF ARCS BETWEEN THE NODES.
- A. NORMAL DATA INCLUDES:
- (1). SIMNUM = 3
 - (2). MINPUT = 3
 - (3). MEANLN = 10
 - (4). NUMOUT = 20
 - (5). INREAD = 0
 - (6). DELADD = 0 IF DELETING ARCS FROM STRUCTURE.
 - DELADD = 1 IF ADDING ARCS TO THE STRUCTURE.
 - (7). ITIME = 50 (IN MILLISECONDS)
 - (8). MMTTR = 30 (IN MINUTES)
 - (9). MEANER = 20
 - (10). MEANIT = 20
 - (11). AVCHNG = 3
 - (12). ASTER = *****
 - (13). N = 30
- B. IF DELADD = 0 INPUT MOST COMPLEX STRUCTURED ADJACENCY MATRIX.
 IF DELADD = 1 INPUT SIMPLEST STRUCTURED ADJACENCY MATRIX.
- C. INPUT 20 (NUMOUT) PAIRS OF VALUES N1 AND N2 WHICH DETERMINE THE ARC TO BE DELETED OR ADDED.
4. TO RUN THE SIMULATION WHICH VARIES THE NUMBER OF LOOPS:
- A. NORMAL DATA INCLUDES:
- (1). SIMNUM = 4
 - (2). MINPUT = 3
 - (3). MEANLN = 10
 - (4). NUMOUT = 20
 - (5). INREAD = 0
 - (6). DELADD = 0 IF DELETING LOOPS FROM STRUCTURE.
 - DELADD = 1 IF ADDING LOOPS TO THE STRUCTURE.
 - (7). ITIME = 50 (IN MILLISECONDS)
 - (8). MMTTR = 30 (IN MINUTES)
 - (9). MEANER = 20
 - (10). MEANIT = 20
 - (11). AVCHNG = 3
 - (12). ASTER = *****
 - (13). N = 30
- B. IF DELADD = 0 INPUT MOST COMPLEX STRUCTURED ADJACENCY MATRIX.
 IF DELADD = 1 INPUT SIMPLEST STRUCTURED ADJACENCY MATRIX.
- C. INPUT 20 (NUMOUT) PAIRS OF VALUES N1 AND N2 WHICH DETERMINE THE LOOP TO BE DELETED OR ADDED.

[illegible]

```

(1). SIMNUM = 5
(2). MINPUT = 1
(3). MEANLN = 10
(4). NUMOUT = 30
(5). INREAD = 0 IF THE NUMBER OF INSTRUCTIONS IN EACH
    ARC IS TO BE DETERMINED RANDOMLY - OTHERWISE
    INREAD = 1
(6). DELADD = 0
(7). ITIME = 50 (IN MILLISECONDS)
(8). MMTTR = 30 (IN MINUTES)
(9). MEANER = 20
(10). MEANIT = 20
(11). AVCHNG = 3
(12). ASTER = *****
(13). N = 30

```

```

B. INPUT ADJACENCY MATRIX
C. IF INREAD = 1 INPUT MATRIX OF ARC LENGTHS.
D. IZ = 1722632 (OR WHATEVER LAST SEED WAS ON PREVIOUS RUN)
NOTE: LAST DATA CARD

```

C. IF INREAD = 1 INPUT MATRIX OF ARC LENGTHS.
D. IZ = 1722632 (OR WHATEVER LAST SEED WAS ON PREVIOUS RUN)
NOTE: LAST DATA CARD

NOTE: LAST DATA CARD

```

COMMON/ALL/N, NODES(30,30)
COMMON/ARRAY/X(30,30), NUMPTS(30), ASTOUT(30), AVCHNG
COMMON/ERROR/ISEED(30,30), ITER(30,30), MEANER, INST, MEANIT, IZ
COMMON/OUT/NSEED, INPUT, SVSEED(30,30)
COMMON/GEN/MEANLN, IX
COMMON/NEW/IW
COMMON/GRAPH/INREAD
DIMENSION TTIME(50), TREP(50), TLINK(50), TKOUNT(50), TINPUT(50)
DIMENSION PSEED(50), F(50), FNA(50), FLA(50)
DIMENSION TSEED(50), TINST(50), PTEST(50)
INTEGER*4 CHANGE, AVCHNG, TESTED(30,30), TEST, SIMNUM, SVSEED
INTEGER*4 DELADD
REAL*4 MTTR
100 FORMAT (I3)
108 FORMAT (I10)
140 FORMAT (I6I5)
151 FORMAT ('1', 13X, 'THIS RUN WILL VARY THE NUMBER OF INSTRUCTIONS B
1ETWEEN EACH NODE.')
152 FORMAT ('1', 13X, 'THIS RUN WILL VARY THE NUMBER OF INPUTS TO BE U
1SEED.')
153 FORMAT ('1', 13X, 'THIS RUN WILL VARY THE NUMBER OF ARCS BETWEEN T
1HE NODES.')
154 FORMAT ('1', 13X, 'THIS RUN WILL VARY THE NUMBER OF LOOPS.').
155 FORMAT ('1', 13X, 'THIS RUN WILL VARY THE RANDOM NUMBER SEED WHICH
1 DETERMINES THE INPUT PATH.')

```

```

156 FORMAT ('0', 13X, 'THE NUMBER OF INSTRUCTIONS IN EACH ARC HAS BEEN
1 READ IN AND IS NOT RANDOM.')
180 FORMAT ('0', 13X, 'ADJACENCY MATRIX'//7X, 30(I4))
203 FORMAT ('0', 13X, 'NOTE: NONEXISTENCE OF AN ARC IS INDICATED BY A
1 ZERO'//)
204 FORMAT ('0', 13X, 'THERE WILL BE', I3, ' GRAPHS EVALUATED THIS RUN
1.'//)
205 FORMAT (7(I5, 5X))
206 FORMAT (A5)
207 FORMAT ('0', 12X, 'INPUT ASSUMPTIONS (VARIABLES):'//13X, 'EXECUTIO
1N TIME PER INSTRUCTION HAS AN EXPONENTIAL DISTRIBUTION WITH A MEAN
2 OF', I3, ' MILLISECONDS.'//13X, 'MEAN TIME TO REPAIR AN ERROR HAS
3AN EXPONENTIAL DISTRIBUTION WITH A MEAN OF', I3, ' MINUTES.'//13X,
4'THE NUMBER OF INPUT PATHS BEING INVESTIGATED IS', I3, '.'//13X,
5'ERRORS ARE SEEDED BY AN EXPONENTIAL DISTRIBUTION WITH A MEAN OF',
6 I3, ' INSTRUCTIONS PER ERROR.'//13X, 'THE AVERAGE NUMBER OF INSTRU
7CTIONS CHANGED WHEN AN ERROR IS FOUND IS', I3, '.')
208 FORMAT ('0', 13X, 'THE NUMBER OF ITERATIONS PER LOOP IS UNIFORMLY
1DISTRIBUTED FROM 1 TO', I3, '.')
209 FORMAT ('0', 13X, 'THE MEAN NUMBER OF INSTRUCTIONS BETWEEN EACH NO
1DE IS EXPONENTIALLY DISTRIBUTED WITH A MEAN OF', I3, '.')
227 FORMAT ('0', 13X, 'SAMPLE INPUT PATH FROM'//13X, 'NODE TO NODE'//)
230 FORMAT('0', 11X, I4, 4X, I4)
240 FORMAT('0', 13X, 'FOR INPUT NUMBER', I3, ' THE EXECUTION TIME IS',
1 F8.2, ' SECONDS WITH A TOTAL REPAIR TIME OF', F8.2, ' HOURS'
2 // 13X, 'AND AN AVERAGE LINK TRAVERSAL TIME OF', F8.2, ' SECONDS')
245 FORMAT ('0', 13X, 'ERROR FOUND'//)
250 FORMAT('1', 13X, I4, ' SEEDED ERRORS REMAINING'//13X, 'MATRIX OF
1SEDED ERRORS REMAINING'//7X, 30(I4))
255 FORMAT(' ', 7X, 30(A4))
260 FORMAT('0', I5, ' ', 30(I4))
265 FORMAT('1', 11X, I3, ' INPUTS'//13X, I4, ' INSTRUCTIONS'//13X,
1I3, ' ERRORS SEEDED'//13X, I3, ' RESIDUAL ERRORS'//13X, F6.2,
2 ' PERCENT RESIDUAL ERRORS'//13X, F6.2, ' PERCENT OF ARCS TESTED'
3 //)
270 FORMAT('0', 9X, F10.5, ' SECONDS EXECUTION TIME'//12X,
1 F10.6, ' HOURS TO REPAIR ERRORS' //12X, F10.6, ' SECONDS AVE
2RAGE ARC TRAVERSAL TIME'//)
275 FORMAT('0', 12X, F5.2, ' IS THE RATIO OF ACTUAL TO MAXIMUM NUMBER
1 OF ARCS'// 13X, F5.2, ' IS THE RATIO OF NODES TO ARCS'//13X,
2 F5.2, ' IS THE RATIO OF LOOPS TO ARCS'//)
280 FORMAT ('0'//13X, 'HISTOGRAM FOR EXECUTION TIME')
281 FORMAT ('0'//13X, 'HISTOGRAM FOR REPAIR TIME')
282 FORMAT ('0'//13X, 'HISTOGRAM FOR PERCENT RESIDUAL ERRORS')
283 FORMAT ('0'//13X, 'HISTOGRAM FOR PERCENT ARCS TESTED')
284 FORMAT ('1', 1X)
285 FORMAT ('0'//13X, 'EXECUTION TIME VS RESIDUAL ERRORS')
286 FORMAT ('0'//13X, 'EXECUTION TIME VS PERCENT RESIDUAL ERRORS')

```

```

287 FORMAT ('0'//13X, 'REPAIR TIME VS RESIDUAL ERRORS')
288 FORMAT ('0'//13X, 'NUMBER OF INPUTS VS RESIDUAL ERRORS')
289 FORMAT ('0'//13X, 'NUMBER OF INSTRUCTIONS VS RESIDUAL ERRORS')
290 FORMAT ('0'//13X, 'RATIO OF ACTUAL TO MAXIMUM NUMBER OF ARCS VS RE
1SIDUAL ERRORS')
291 FORMAT ('0'//13X, 'RATIO OF NODES TO ARCS VS RESIDUAL ERRORS')
292 FORMAT ('0'//13X, 'RATIO OF LOOPS TO ARCS VS RESIDUAL ERRORS')
293 FORMAT ('0'//13X, 'EXECUTION TIME VS RATIO OF ACTUAL TO MAXIMUM NUM
1BER OF ARCS')
294 FORMAT ('0'//13X, 'EXECUTION TIME VS RATIO OF NODES TO ARCS')
295 FORMAT ('0'//13X, 'EXECUTION TIME VS RATIO OF LOOPS TO ARCS')
296 FORMAT ('0'//13X, 'NUMBER OF INPUTS VS PERCENT OF ARCS TESTED')
297 FORMAT ('0'//13X, 'RATIO OF ACTUAL TO MAXIMUM NUMBER OF ARCS VS PE
1RCENT OF ARCS TESTED')
298 FORMAT ('0'//13X, 'RATIO OF NODES TO ARCS VS PERCENT OF ARCS TESTE
1D')
299 FORMAT ('0'//13X, 'RATIO OF LOOPS TO ARCS VS PERCENT OF ARCS TESTE
1D')
300 FORMAT ('0'//13X, 'NUMBER OF INPUTS VS EXECUTION TIME')
301 FORMAT ('0'//13X, 'NUMBER OF INPUTS VS PERCENT RESIDUAL ERRORS')
302 FORMAT ('0'//13X, 'EXECUTION TIME VS NUMBER OF INSTRUCTIONS')
303 FORMAT ('1', 59X, 'DATA SUMMARY '//62X, 'PERCENT', 3X, 'PERCENT',
1 2X, 'EXECUTION', 3X, 'REPAIR', 4X, 'ACTUAL', 4X, 'NODES', 5X,
2 'LOOPS', 4X, 'RUN', 35X, 'ERRORS', 3X, 'RESIDUAL', 2X, 'RESIDUAL',
3 4X, 'ARCS', 6X, 'TIME', 6X, 'TIME', 5X, 'TO MAX', 6X, 'TO', 8X,
4 'TO', 2X,
5 'NUMBER', 5X, 'NODES', 4X, 'INPUTS', 4X, 'INSTR.', 4X, 'SEEDS',
6 4X, 'ERRORS', 4X, 'ERRORS', 4X, 'TESTED', 4X, ' (SEC)', 7X,
7 ' (HRS)', 4X, 'ARCS', 6X, 'ARCS', 6X, 'ARCS')
304 FORMAT ('0', 2X, I3, 1X, '*', 5X, I3, 7X, F5.0, 3(5X, F5.0),
1 2(4X, F6.2), 2X, 2F10.5, 3X, F5.3, 2(5X, F5.3))
305 FORMAT ('0', 13X, 'THE LAST SEED USED WAS', I11, '.')
C READ IN THE TYPE OF SIMULATION TO BE RUN (SIMNUM)
C READ (5,100) SIMNUM
C READ IN NUMBER OF INPUTS TO BE USED (MINPUT)
C READ (5,100) MINPUT
C READ IN THE MEAN NUMBER OF INSTRUCTIONS BETWEEN EACH NODE (MEANLN)
C READ (5,100) MEANLN
C READ IN THE NUMBER OF GRAPHS TO BE EVALUATED (NUMOUT)
C READ (5,100) NUMOUT
C READ IN WHETHER OR NOT THE NUMBER OF INSTRUCTIONS IS TO BE READ IN
C READ (5,100) INREAD
C READ IN WHETHER DELETING AN ARC FROM THE STRUCTURE OR
C ADDING AN ARC TO THE STRUCTURE.
C READ (5,100) DELADD
C READ IN THE MEAN TIME TO EXECUTE AN INSTRUCTION IN MILLISECONDS
C (ITIME)
C READ (5,100) ITIME

```

```

C      READ IN MEAN TIME TO REPAIR AN ERROR IN MIN (MMTTR)
C      READ (5,100) MMTTR
C      READ IN MEAN NUMBER OF INSTRUCTIONS THAT ARE ERROR FREE (MEANER)
C      READ (5,100) MEANER
C      READ IN MEAN NUMBER OF ITERATIONS FOR EACH LOOP (MEANIT)
C      READ (5,100) MEANIT
C      READ IN MEAN NUMBER OF INSTRUCTIONS CHANGED WHEN AN ERROR IS
C      FOUND (AVCHNG)
C      READ (5,100) AVCHNG
C      READ IN THE ASTERICKS TO PUT AROUND OUTPUT ARRAYS (ASTER)
C      READ (5,206) ASTER
C      READ IN THE NUMBER OF NODES (N)
C      READ (5,100) N
C      READ IN THE ADJACENCY MATRIX (NODES)
      DO 20 I=1,N
20    READ (5,140) (NODES(I,J), J=1,N)
      IF (SIMNUM.EQ.1) WRITE (6,151)
      IF (SIMNUM.EQ.2) WRITE (6,152)
      IF (SIMNUM.EQ.3) WRITE (6,153)
      IF (SIMNUM.EQ.4) WRITE (6,154)
      IF (SIMNUM.EQ.5) WRITE (6,155)
      IF (INREAD.EQ.1) WRITE(6,156)
      WRITE (6,207) ITIME, MMTTR, MINPUT, MEANER, AVCHNG
      WRITE (6,208) MEANIT
      WRITE (6,209) MEANLN
      WRITE (6,204) NUMOUT
      NNOUT = 1
      IX = 71286223
      IW = IX
C      DETERMINE THE MAXIMUM NUMBER OF ARCS IN THE ADJACENCY MATRIX
      MAXARC = N*(N-1)
C      NUMPTS IS THE LABEL FOR THE OUTPUT ARRAY
      DO 57 I = 1,N
      NUMPTS(I) = I
C      ASTER AND ASTOUT ARE TO PUT ASTERICKS AROUND THE ARRAY
      ASTOUT(I) = ASTER
57    CONTINUE
      CALL DIGRAF
59    DO 62 I = 1,N
      DO 61 J = 1,N
C      ISEED IS THE ARRAY OF ERRORS SEEDED
      ISEED(I,J) = 0
61    CONTINUE
62    CONTINUE
C      SEED THE PROGRAM WITH ERRORS
      IF (SIMNUM.EQ.1) IX = 1722632
      CALL SEED
C      THIS SEED CAN BE CHANGED TO THE LAST VALUE OF THE SEED ON THE

```

```

C      PREVIOUS TEST IF A CONTINUATION OF THE DATA IS DESIRED.
      IF (SIMNUM.EQ.5) READ (5,108) IX
63  ARCS = 0.
      INST = 0
      NSEED = 0
      DO 65 I=1,N
      DO 65 J=1,N
C      ZERO THE MATRIX WHICH RECORDS WHETHER OR NOT AN ARC HAS BEEN TEST
      TESTED(I,J) = 0
      IF (NODES(I,J).NE.1) GO TO 65
C      CALCULATE THE NUMBER OF ARCS IN THE PROGRAM
      ARCS = ARCS + 1.
C      CALCULATE THE NUMBER OF INSTRUCTIONS
      INST = INST + X(I,J)
C      CALCULATE THE NUMBER OF ERRORS SEEDED
      NSEED = NSEED + ISEED(I,J)
65  CONTINUE
C      CALCULATE THE NUMBER OF LOOPS IN THE PROGRAM
      LCCPS = 0
      DO 68 I=1,N
      DO 68 J=1,I
      IF (NODES(I,J).EQ.1) LOOPS = LOOPS + 1
68  CONTINUE
C      CHOOSE SAMPLE INPUT
      INPUT = 1
77  TCTREP = 0.
      TIME = 0.
      AVLINK = 0.
      TTIME(NNOUT) = 0.
      TREP(NNOUT) = 0.
      TLINK(NNOUT) = 0.
78  IZ = IX
      NODE = 1
      WRITE (6,227)
C      NUMSUC IS THE NUMBER OF SUCCESSORS
79  NUMSUC = 0
      DO 80 J=1,N
      IF (NODES(NODE,J).EQ.1) NUMSUC = NUMSUC + 1
80  CONTINUE
      IF (NUMSUC.EQ.0) GO TO 96
      IF (NUMSUC.NE.1) GO TO 82
      K = 1
      GO TO 83
C      THE SUCCESSORS ARE CHOSEN RANDOMLY WITH A UNIFORM DISTRIBUTION
82  CALL RANDOM(IZ, U, 1)
      K = 1 + NUMSUC*U
83  KK = 0
      DO 85 J = 1,N

```

```

      IF (NODES(NODE,J).EQ.1) KK = KK + 1
      IF (KK.EQ.K) GO TO 86
85  CONTINUE
86  L = J
C   TIME IS ASSUMED TO HAVE A MEAN OF ITIME MILLISECONDS PER
C   INSTRUCTION WITH AN EXPONENTIAL DISTRIBUTION
      CALL EXPON(IZ, XTIME, 1)
      XTIME = XTIME * ITIME
      IF (NODE.LE.L) GO TO 88
C   CONVERT TIME INTO HOURS
      TIME = TIME + XTIME * ITER(NODE,L) * (X(NODE,L) + X(L,NODE)) /
1  1.0E03
      GO TO 89
88  TIME = TIME + X(NODE,L) * XTIME/ 1.0E03
89  WRITE (6,230) NODE,L
      TESTED(NODE,L) = 1
C   IS THERE AN ERROR IN THIS PATH
      IF (ISEED(NODE,L).EQ.0) GO TO 95
C   ERROR FOUND
      WRITE (6,245)
C   SEE IF A NEW ERROR SHOULD BE INSERTED
      CALL NUSEED
90  ISEED(NODE,L) = ISEED(NODE,L) - 1
      CALL EXPON(IZ, U, 1)
C   MEAN TIME TO REPAIR HAS AN EXPONENTIAL DISTRIBUTION WITH A
C   MEAN OF MMTTR MINUTES
      MTTR = MMTTR * U
C   TOTREP IS THE TOTAL REPAIR TIME IN HOURS
      TOTREP = TOTREP + MTTR/ 60.
      GO TO 78
95  NODE = L
      GO TO 79
C   AVLINK IS THE AVERAGE ARC TRAVERSAL TIME IN MINUTES
96  AVLINK = (TIME/ARCS)
      WRITE (6,240) INPUT, TIME, TOTREP, AVLINK
      IF (SIMNUM.EQ.2) GO TO 97
      TTIME(NNOUT) = TTIME(NNOUT) + TIME
      TREP(NNOUT) = TREP(NNOUT) + TOTREP
      TLINK(NNOUT) = TLINK(NNOUT) + AVLINK
      GO TO 99
97  NOUT = NNOUT - 1
      IF (NNOUT.GT.1) GO TO 98
      TLINK(1) = AVLINK
      TREP(1) = TOTREP
      TTIME(1) = TIME
      GO TO 99
98  TLINK(NNOUT) = TLINK(NOUT) + AVLINK
      TREP(NNOUT) = TREP(NOUT) + TOTREP

```

```

    TTIME(NNOUT) = TTIME(NOUT) + TIME
99  TIME = 0.
    TOTREP = 0.
    AVLINK = 0.
    INPUT = INPUT + 1
C    KCOUNT IS A COUNTER OF THE NUMBER OF SEEDED ERRORS REMAINING
    KCOUNT = 0
    DO 101 I = 1,N
    DO 101 J = 1,N
    IF (NODES(I,J).EQ.1) KCOUNT = KCOUNT + ISEED(I,J)
101  CONTINUE
    WRITE (6,250) KCOUNT, (NUMPTS(I), I = 1,N)
    WRITE (6,255) (ASTOUT(I), I = 1,N)
    DO 110 I = 1,N
C 110  WRITE (6,260) NUMPTS(I), (ISEED(I,J), J=1,N)
    DISTINCT SEED FOR RANDOM NUMBER GENERATOR FOR EACH INPUT
    IX = IX - 12345
    IF (SIMNUM.EQ.2) GO TO 111
    IF (SIMNUM.EQ.5) GO TO 111
C    MINPUT IS THE MAXIMUM NUMBER OF INPUT PATHS TO BE CHECKED
    IF (INPUT.LE.MINPUT) GO TO 78
C    CALCULATE THE NUMBER OF ARCS TESTED
111  TEST = 0
    DO 112 I = 1,N
    DO 112 J = 1,N
    IF (TESTED(I,J).EQ.1) TEST = TEST + 1
112  CONTINUE
    INPUT = INPUT - 1
C    PTEST IS THE PERCENT OF ARCS TESTED
    PTEST(NNOUT) = 100. * TEST / ARCS
C    F IS THE RATIO OF ACTUAL TO MAXIMUM NUMBER OF ARCS
    F(NNOUT) = ARCS / MAXARC
C    FNA IS THE RATIO OF NODES TO ARCS
    FNA(NNOUT) = N / ARCS
C    FLA IS THE RATIO OF LOOPS TO ARCS
    FLA(NNOUT) = LOOPS / ARCS
    TINST(NNOUT) = INST
    TINPUT(NNOUT) = INPUT
    TSEED(NNOUT) = NSEED
    TKOUNT(NNOUT) = KCOUNT
    PSEED(NNOUT) = 100. * (TKOUNT(NNOUT) / NSEED)
    WRITE (6,265) INPUT, INST, NSEED, KCOUNT, PSEED(NNOUT),
1  PTEST(NNOUT)
    WRITE (6,270) TTIME(NNOUT), TREP(NNOUT), TLINK(NNOUT)
    WRITE (6,275) F(NNOUT), FNA(NNOUT), FLA(NNOUT)
    NNOUT = NNOUT + 1
    IF (NNOUT.GT.NUMOUT) GO TO 118
    IF (SIMNUM.EQ.1) MEANLN = MEANLN + 1

```

```

      IF (SIMNUM.EQ.1) GO TO 57
      IF (SIMNUM.EQ.2) INPUT = INPUT + 1
      IF (SIMNUM.EQ.2) GO TO 78
      DO 113 I = 1,N
      DO 113 J = 1,N
113  ISEED(I,J) = SVSEED(I,J)
      IF (SIMNUM.EQ.5) GO TO 63
115  IF (DELADD.EQ.0) CALL DELARC
      IF (DELADD.EQ.1) CALL ADDARC
116  WRITE (6,180) (NUMPTS(I), I=1,N)
      WRITE (6,255) (ASTOUT(I), I = 1,N)
      DO 117 I = 1,N
117  WRITE (6,260) NUMPTS(I), (NODES(I,J), J=1,N)
      WRITE (6,203)
      GO TO 63
118  WRITE (6,305) IZ
      WRITE (6,284)
      CALL PLOTP (TTIME, TKOUNT, NUMOUT, 0)
      WRITE (6,285)
      WRITE (6,284)
      CALL PLOTP (TTIME, PSEED, NUMOUT, 0)
      WRITE (6,286)
      WRITE (6,284)
      CALL PLOTP (TREP, TKOUNT, NUMOUT, 0)
      WRITE (6,287)
      IF (SIMNUM.NE.2) GO TO 119
      WRITE (6,284)
      CALL PLOTP (TINPUT, TKOUNT, NUMOUT, 0)
      WRITE (6,288)
      WRITE (6,284)
      CALL PLOTP (TINPUT, PTEST, NUMOUT, 0)
      WRITE (6,296)
      WRITE (6,284)
      CALL PLOTP (TINPUT, TTIME, NUMOUT, 0)
      WRITE (6,300)
      WRITE (6,284)
      CALL PLOTP (TINPUT, PSEED, NUMOUT, 0)
      WRITE (6,301)
119  IF ((SIMNUM.EQ.2).OR.(SIMNUM.EQ.5)) GO TO 121
      WRITE (6,284)
      CALL PLOTP (TTIME, TINST, NUMOUT, 0)
      WRITE (6,302)
      WRITE (6,284)
      CALL PLOTP (TINST, TKOUNT, NUMOUT, 0)
      WRITE (6,289)
      IF (SIMNUM.EQ.1) GO TO 121
      WRITE (6,284)
      CALL PLOTP (F, TKOUNT, NUMOUT, 0)

```

```

WRITE (6,290)
WRITE (6,284)
CALL PLOTP (FNA,      TKOUNT, NUMOUT, 0)
WRITE (6,291)
WRITE (6,284)
CALL PLOTP (FLA,      TKOUNT, NUMOUT, 0)
WRITE (6,292)
WRITE (6,284)
CALL PLOTP (TTIME,  F,      NUMOUT, 0)
WRITE (6,293)
WRITE (6,284)
CALL PLOTP (TTIME,  FNA,    NUMOUT, 0)
WRITE (6,294)
WRITE (6,284)
CALL PLOTP (TTIME,  FLA,    NUMOUT, 0)
WRITE (6,295)
WRITE (6,284)
CALL PLOTP (F,      PTEST,  NUMOUT, 0)
WRITE (6,297)
WRITE (6,284)
CALL PLOTP (FNA,    PTEST,  NUMOUT, 0)
WRITE (6,298)
WRITE (6,284)
CALL PLOTP (FLA,    PTEST,  NUMOUT, 0)
WRITE (6,299)
121 DO 125 K = 1,2
    WRITE (6,303)
    WRITE (6,255) (ASTOUT(I), I = 1,N)
    DO 122 I = 1,NUMOUT
122 WRITE (6,304) I, N, TINPUT(I), TINST(I), TSEED(I), TKOUNT(I),
1 PSEED(I), PTEST(I), TTIME(I), TREP(I), F(I), FNA(I), FLA(I)
125 CONTINUE
    IF ((SIMNUM.NE.3).AND.(SIMNUM.NE.4)) CALL GRAPHO
    C   NEED A MINIMUM OF 10 GRAPHS TO ANALYZE BEFORE CALLING HISTG
    IF (SIMNUM.NE.5) GO TO 126
    CALL HISTG (TTIME, NUMOUT, 0)
    WRITE (6,280)
    CALL HISTG (TREP,  NUMOUT, 0)
    WRITE (6,281)
    CALL HISTG (PSEED, NUMOUT, 0)
    WRITE (6,282)
    CALL HISTG (PTEST, NUMOUT, 0)
    WRITE (6,283)
126 CONTINUE
    C   STOP
    C   END

```



```

C      SPREAD ERRORS TO GIVE MEAN OF MEANER
C
DO 60 I = 1,N
IER(I) = MEANER * ER(I)
60 CONTINUE
II = 1
C      NSEED IS THE NUMBER OF ERRORS SEEDED
NSEED = 0
C      INST IS THE NUMBER OF INSTRUCTIONS
INST = 0
C      NUMBER IS THE ERROR SEED/INSTRUCTION COMPARATOR
NUMBER = 0
DO 70 I = 1,N
DO 65 J = 1,N
IF (NODES(I,J).EQ.0) GO TO 65
C      COUNT THE NUMBER OF INSTRUCTIONS
INST = INST + X(I,J)
C      LOOPS ARE ASSUMED TO HAVE AN ERROR RATE PROPORTIONAL TO THE
C      NUMBER OF INSTRUCTIONS IN THE LOOP AND THE NUMBER OF ITERATIONS
C      THROUGH THE LOOP
IF (J.LE.I) IER(II) = IER(II) / ( (X(I,J) * ITER(I,J))/MEANER)
63 NUMBER = NUMBER + IER(II)
IF (INST.LT.NUMBER) GO TO 64
ISEED(I,J) = ISEED(I,J) + 1
NSEED = NSEED + 1
II = II + 1
IF (II.LE.N) GO TO 63
II = 1
GO TO 63
64 NUMBER = NUMBER - IER(II)
65 CONTINUE
70 CONTINUE
DO 71 I = 1,N
DO 71 J = 1,N
C      SVSEED IS THE SAVE COPY OF THE ORIGINAL SEEDED ERRORS
71 SVSEED(I,J) = ISEED(I,J)
C      NSEED IS THE NUMBER OF ERRORS SEEDED
WRITE (6,210) INST
WRITE (6,205) (IER(I), I = 1,N)
WRITE (6,220) NSEED, (NUMPTS(I), I = 1,N)
WRITE (6,222) (ASTOUT(I), I=1,N)
DO 75 I = 1,N
WRITE (6,225) NUMPTS(I), (ISEED(I,J), J=1,N)
75 CONTINUE
C
C      RETURN
C
C      END

```

```

SUBROUTINE NUSEED
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C   CORRECTING PREVIOUS ERROR MAY CREATE A NEW ERROR
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
COMMON/ALLL/N, NODES(30,30)
COMMON/ARRAY/X(30,30), NUMPTS(30), ASTOUT(30), AVCHNG
COMMON/ERROR/ISEED(30,30), ITER(30,30), MEANER, INST, MEANIT, IZ
COMMON/OUT/NSEED, INPUT, SVSEED(30,30)
COMMON/NEW/IW
INTEGER*4 CHANGE, AVCHNG
DIMENSION ERR(2)
247 FORMAT ('0', 13X, 'NEW ERROR INSERTED FROM NODE', I4, ' TO NODE',
1 I4//)
NUM = 0
C   THE ERROR IS EQUALLY LIKELY TO RESULT IN CHANGES TO ALL ARCS
CALL RANDOM (IW, ERR, 2)
I = 1 + N*ERR(1)
J = 1 + N*ERR(2)
C   NEW ERROR CREATED AT (I,J)
C   IS THIS A NODE OF THE DIRECTED GRAPH
10 IF (NODES(I,J).EQ.1) GO TO 20
J = J + 1
IF (J.GT.N) J = 1
NUM = NUM + 1
C   IF NODE I IS A TERMINAL NODE NO NEW ERROR IS INSERTED
IF (NUM.GT.N) RETURN
GO TO 10
C   CHANCE OF INSERTING A NEW ERROR IS BASED ON THE NUMBER OF
C   INSTRUCTIONS THAT MUST BE CHANGED
20 CALL RANDOM (IW, ERNU, 1)
AVCHNG IS THE ASSUMED (INPUT) AVERAGE NUMBER OF INSTRUCTIONS
CHANGED WHEN AN ERROR IS FOUND
CHANGE IS THE UNIFORMLY DISTRIBUTED NUMBER OF INSTRUCTIONS CHANGED
CHANGE = 1 + AVCHNG*ERNU*2.
C   CRATIO IS THE RATIO OF INSTRUCTIONS CHANGED TO THE NUMBER OF
C   INSTRUCTIONS IN THE UNIFORMLY RANDOMLY CHOSEN ARC
CRATIO = CHANGE / X(I,J)
C   THE CRITERIA FOR INSERTING A NEW ERROR IS UNIFORMLY DISTRIBUTED
CALL RANDOM (IW, RATIO, 1)
C   IF THE CHANGE RATIO (CRATIO) IS GREATER THAN THE CRITERIA (RATIO),
C   THEN A NEW ERROR IS INSERTED
IF (CRATIO.LE.RATIO) RETURN
50 ISEED(I,J) = ISEED(I,J) + 1
NSEED = NSEED + 1
WRITE (6,247) I,J
RETURN

```



```

C      KK IS THE LAST NODE IN THE GIVEN LEVEL
      KK = M
      KT = KK + 1
      IF (NODES(1,KT).EQ.0) GO TO 6
5     CONTINUE
C      KM IS THE LAST NODE IN THE NEXT LEVEL
6     DO 7 M = KK,N
      IF (NODES(KK,M).EQ.0) GO TO 7
      KM = M
      KX = KM + 1
      IF (NODES(KK,KX).EQ.0) GO TO 8
7     CONTINUE
8     DO 50 I = 2,NN
      II = I - 1
C      NUMSUC IS THE NUMBER OF SUCCESSORS
      NUMSUC = 0
      DO 20 J = I,KM
      IF (NODES(I,J).EQ.0) GO TO 20
      DO 18 K = 1,II
C      IS THERE A NODE ALREADY IN EXISTENCE
      IF (ND(K,J).EQ.0) GO TO 18
C      CONNECT NEW NODE TERMINATING AT EXISTING NODE
17     XX(1) = X(I)
      XX(2) = X(J)
      YY(1) = Y(I)
      YY(2) = Y(J)
      CALL DRAWP(2,XX,YY,ITB,RTB)
      ND(I,J) = 1
      GO TO 20
18     CONTINUE
      NUMSUC = NUMSUC + 1
20     CONTINUE
      IF (NUMSUC.LE.0) GO TO 44
      IF (NUMSUC.NE.1) GO TO 30
C      FANOUT TO ONE NEW NODE
      L = LABEL + 1
      LABEL = LABEL + 1
      ND(I,L) = 1
      X(L) = X(I)
      Y(L) = Y(I) - CONST
      XX(1) = X(I)
      XX(2) = X(L)
      YY(1) = Y(I)
      YY(2) = Y(L)
      CALL DRAWP(2,XX,YY,ITB,RTB)
C      IS THERE A NODE ABOVE THAT NEEDS CONNECTING TO THIS NODE
      DO 28 K = 1,II
      IF (ND(K,L).EQ.1) GO TO 28

```

```

      IF (NODES(K,L).EQ.0) GO TO 28
      XX(1) = X(K)
      YY(1) = Y(K)
      XX(2) = X(L)
      YY(2) = Y(L)
      ND(K,L) = 1
      CALL DRAWP(2,XX,YY,ITB,RTB)
28  CONTINUE
      GO TO 44
C   FANOUT TO MULTIPLE NODES
30  FAN = 2.*XCONST / (NUMSUC - 1)
      DO 35 L=1,NUMSUC
      M = LABEL + L
      ND(I,M) = 1
      IF (L.GT.1) GO TO 31
      X(M) = X(I) - XCONST
      GO TO 33
31  LM = M - 1
      X(M) = X(LM) + FAN
33  Y(M) = Y(I) - CONST
      XX(1) = X(I)
      YY(1) = Y(I)
      XX(2) = X(M)
      YY(2) = Y(M)
      CALL DRAWP(2,XX,YY,ITB,RTB)
C   IS THERE A NODE ABOVE THAT NEEDS CONNECTING TO THIS NODE
      DO 34 K=1,II
      IF (ND(K,M).EQ.1) GO TO 34
      IF (NODES(K,M).EQ.0) GO TO 34
      XX(1) = X(K)
      YY(1) = Y(K)
      XX(2) = X(M)
      YY(2) = Y(M)
      ND(K,M) = 1
      CALL DRAWP(2,XX,YY,ITB,RTB)
34  CONTINUE
35  CONTINUE
      LABEL = LABEL + NUMSUC
      KOUNT = KOUNT + 1
40  IF (KOUNT.LT.2) GO TO 44
C   REDUCE THE X-DIRECTION SCALE FACTOR
      XCONST = XCONST/1.5
      KOUNT = 0
C   IS THIS NODE STILL ON THE GIVEN LEVEL
44  IF (I.LT.KK) GO TO 50
C   NO, CHANGE LEVELS
      LEVEL = LEVEL + 1
C   DETERMINE THE LAST NODE IN THE NEW LEVEL

```

```

DO 45 M = 1,N
IF (NODES(I,M).EQ.0) GO TO 45
KK = M
KT = KK + 1
IF (NODES(I,KT).EQ.0) GO TO 46
C 45 CONTINUE
DETERMINE THE LAST NODE IN THE NEXT LEVEL AFTER THE PRESENT LEVEL
C 46 DO 47 M = KK,N
IF (NODES(KK,M).EQ.0) GO TO 47
KM = M
KX = KM + 1
IF (NODES(KK,KX).EQ.0) GO TO 50
C 47 CONTINUE
C 50 CONTINUE
GRAPH LOOPS
DO 90 I = 2,N
K = I - 1
DO 80 J = 1,K
IF (NODES(I,J).EQ.0) GO TO 80
PI = 3.1415926
C ESTABLISH THE DIRECTION OF CURVE FOR LOOP
IF (X(I).LT.X(J)) PI = -PI
C STARTING POINT FOR CURVE
XL(1) = X(I)
YL(1) = Y(I)
C ENDING POINT FOR THE CURVE
XL(36) = X(J)
YL(36) = Y(J)
C XDIF AND YDIF ARE THE DIFFERENCE BETWEEN THE X AND Y VALUES RESP.
XDIF = X(I) - X(J)
YDIF = Y(I) - Y(J)
C R IS THE RADIUS OF THE LOOP
R = 0.5 * SQRT(XDIF**2 + YDIF**2)
C (XC,YC) IS THE CENTER OF THE LOOP
XC = (X(I) + X(J))/2.
YC = (Y(I) + Y(J))/2.
C THETE(1) IS THE INITIAL ANGLE
THETA(1) = ATAN2(YDIF,XDIF)
C CREATE 36 POINTS IN A SEMICIRCLE FORMING THE LOOP
DO 70 L = 2,35
M = L - 1
THETA(L) = THETA(M) + PI/36.
XL(L) = R * COS (THETA(L)) + XC
YL(L) = R * SIN (THETA(L)) + YC
70 CONTINUE
CALL DRAWP(36,XL,YL,ITB,RTB)
80 CONTINUE
90 CONTINUE

```


LIST OF REFERENCES

1. Boehm, B. W., "Software and Its Impact: A Quantitative Assessment", Datamation, v. , p. 48-59, May 1973.
2. Boehm, B. W., "The High Cost of Software", Proceedings of a Symposium on the High Cost of Software, v. Jack Goldberg (ed), Stanford Research Institute, p. 27-40, 1973.
3. Bradley, G. H., Green, T. F., Howard, G. T., and Schneidewind, N. f., "Structure and Error Detection in Computer Software", Proceedings of the American Institute of Industrial Engineers National Meeting, v. 1, p. 1-6, May 21, 1975.
4. Schneidewind, N. F. , "An Approach to Software Reliability Prediction and Quality Control", AFIPS Conference Proceedings, v. 41, Part II, Fall Joint Computer Conference, p. 837-838, 1972.
5. Baker, F. T., "Chief Programmer Team Management of Production Programming", IBM Systems Journal, v. 11, No. 1, p. 65-66, 1972.
6. Rizza, J. B. and Hacker, D., "Quality Assurance Inspection and Test Tools - An Application", Proceedings of a Workshop on Currently Available Program Testing Tools, v. 1, p. 9-10, April 1975.
7. Howden, W. E., "Systems for Automating the Generation of Program Test Data", Proceedings of a Workshop on Currently Available Program Testing Tools, v. 1, p. 37-39, April 1975.

8. Von Alven, W. H., (ed), Reliability Engineering, p. 155-156, ARINC Research Corporation, Prentice Hall, 1964.
9. Stucki, L. G., "Automatic Generation of Self-Metric Software", Record of 1973 IEEE Symposium on Computer Software Reliability, v. New York City, p. 94-100, April 30 - May 2, 1973.
10. McCormick, J. M. and Salvadori, M. G., Numerical Methods in FORTRAN, p. 290-295, Prentice Hall, 1964.

INITIAL DISTRIBUTION LIST

	No.	Copies
1. Defense Documentation Center Cameron Station Alexandria, Virginia 22314	2	
2. Library, Code 0212 Naval Postgraduate School Monterey, California 93940	2	
3. Department Chairman, Code 72 Computer Science Group Naval Postgraduate School Monterey, California 93940	1	
4. Professor Norman F. Schneidewind, Code 32B Department of Operations Research and Administrative Sciences Naval Postgraduate School Monterey, California 93940	1	
5. LT Thomas F. Green, USN 3246 Fenelon Street San Diego, California 92106	1	

14 JAN 76

22 OCT 76

24044

23616

160963

Thesis

G742

Green

c.2

Software error de-
tection model.

14 JAN 76

22 OCT 76

24044

23616

160963

Thesis

G742

Green

c.2

Software error de-
tection model.

thesG742

Software error detection model.



3 2768 002 13876 0

DUDLEY KNOX LIBRARY